FAST OBJECT DETECTION

BY

MOHAMMAD AMIN SADEGHI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Doctoral Committee:

Associate Professor Derek Hoiem, Chair
Professor David Forsyth, Director of Research
Associate Professor Deva Ramanan, Carnegie Mellon University
Associate Professor Svetlana Lazebnik
Assistant Professor Mani Golparvar-fard

Abstract

The ultimate goal of computer vision is to understand images. We describe methods to understand images at two levels. One is at the level of description of images which we produce using sentences. These sentences talk about the things that are present in the image and about where they are and what they are doing. Then we ask in what ways should we describe images. We introduce visual phrases that are composite chunks of meaning. We show that object detectors could be better at detecting some visual phrases than detecting single objects.

This process of image understanding needs to use a lot of detectors. Running conventional object detectors at the rate required for image understanding could be very slow. We study fast object detection from an engineering perspective. We argue that a desirable object detector must: (1) be able to work with legacy templates; (2) be random access; (3) be able to trade accuracy versus speed; (4) have any-time property. We describe a method to have all of these features together for a fast detector. We apply these techniques to deformable parts model object detectors and show two orders of magnitude speed-up while adding their desirable features. We finally investigate the consequences of this architecture with a view of improving convolutional neural networks.

To Wife, Father and Mother.

Acknowledgments

During my five years of research that lead to this thesis, by far the greatest support came from my adviser, Professor David Forsyth. Without his extraordinary support, this work would have never been successful. I also thank Ali Farhadi for his help and guidance before and during my PhD.

Also thanks to my committee members, Derek Hoiem, Deva Ramanan, Svetlana Lazebnik and Mani Golparvar. And finally, thanks to my wife, parents, and great friends who endured this long process with me, always offering support and love.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The ultimate goal of computer vision is to understand images. Specifically, we wan to understand images (1) more accurately, (2) with greater details and (3) faster. In this dissertation we explore sentence generation and visual phrases that help understand images with greater details. We show that in some cases visual phrases help us understand images more accurately as well. We finally discuss techniques to detect object faster.

Sentences are one way to express images in a human readable format. We illustrate a framework to automatically describe images with sentences [1]. This framework maps images and sentences into a joint representation. It then establishes a mapping from images to sentences (or vice versa). Follow-up work has built upon this framework and proven its effectiveness.

In this work we used then state-of-the-art object detectors and scene descriptors as input features. This work was a demonstration on on how well a computer can understand a scene and communicate in human language. We noticed that object detectors miss certain aspects of meaning. We asked in what way should we describe images.

We explored composites of objects and called them "Visual Phrases" [2]. Our visual phrases were initially made of either two objects with some relation (e.g. "a person riding a horse") or an action imposed on an object (e.g. "dog running"). We showed that object detectors are better at detecting some visual phrases that individual objects. For some visual phrases this improvement was very significant. For example, the average precision for detecting "a horse and a rider jumping" jointly was 60% higher that detecting the individual objects and then reasoning about their relations.

We showed that a decoding procedure is helpful to reason about the final outputs. This helped us beat then state-of-the-art in PASCAL detection accuracy by benefiting from context and redundancy. We also developed a dataset of visual phrases that is still being used by vision researchers. This work introduced new questions and challenges. For example, as visual phrases become more detailed there are fewer examples available for training so training becomes more difficult. We studied visual phrases in increasing granularities [3]. We showed improvement in detection accuracy depends on the visual phrase and the right granularity must be chosen opportunisticly.

This process of image understanding needs to use a lot of detectors. Running conventional object detectors at the rate required for image understanding could be very slow. We study a wide range of techniques to speed-up template evaluation and produce an order of magnitude speed-up [4]. Furthermore, we study techniques for fast object detection from an engineering perspective. We argue that a desirable object detector must: 1- be able to work with legacy templates, 2- be random access, 3- be able to trade accuracy versus speed, 4- have any-time property. We describe a method to have all of these features together for a fast detector. We apply these techniques to deformable parts model and show that fast object detectors with these characteristics are possible [5].

Our fast object detection algorithm relies on the combination of several speed-up techniques. These algorithms include cascades, vector quantization and hardware optimizations. These techniques are generally applicable to convolutional neural networks as well. Finally, we investigate the effect of several design decisions on the performance of vector quantization.

Each chapter in this dissertation comes with its own introduction and literature review. We discuss describing images with sentences in chapter 2. We discuss visual phrases in Chapter 3 as a tool to optimize the use of object detectors for image understanding. We study fast object detection techniques in 4. We finally study optimal design decisions for fast object detection in Chapter 5.

Chapter 2

Interpreting Images With Sentences

2.1 Introduction

For most pictures, humans can prepare a concise description in the form of a sentence relatively easily. Such descriptions might identify the most interesting objects, what they are doing, and where this is happening. These descriptions are rich, because they are in sentence form. They are accurate, with good agreement between annotators. They are concise: much is omitted, because humans tend not to mention objects or events that they judge to be less significant. Finally, they are consistent: in our data, annotators tend to agree on what is mentioned. Barnard *et al.* name two applications for methods that link text and images: **Illustration**, where one finds pictures suggested by text (perhaps to suggest illustrations from a collection); and **annotation**, where one finds text annotations for images (perhaps to allow keyword search to find more images) [6].

We investigated methods to generate short descriptive sentences from images. We introduced a dataset to study this problem (section 2.3.1). We introduced a simple representation intermediate between images and sentences (section 2.2.1). We described a novel, discriminative approach that produces very good results at sentence annotation (section 2.2.4). For illustration, out of vocabulary words pose serious difficulties, and we show methods to use distributional semantics to cope with these issues (section 2.3.4).

Evaluating sentence generation is very difficult, because sentences are fluid, and quite different sentences can describe the same phenomena. Worse, synecdoche (for example, substituting "animal" for "cat" or "bicycle" for "vehicle") and the general richness of vocabulary means that

many different words can quite legitimately be used to describe the same picture. In section 2.3, we describe a quantitative evaluation of sentence generation at a useful scale.

Linking individual words to images has a rich history. Here we cover papers published before this work in 2010. In the last section of this chapter we will cover the latest and the most prominent follow-up work.

The first image annotation system is due to Mori *et al.* [7]; Duygulu *et al.* continued this tradition using models from machine translation [8]. Since then, a wide range of models has been deployed (reviews in [9, 10]); the current best performer is a form of nearest neighbours matching [11]. The most recent methods perform fairly well, but still find difficulty **placing** annotations on the correct regions.

Sentences are richer than lists of words, because they describe activities, properties of objects, and relations between entities (among other things). Such relations are revealing: Gupta and Davis show that respecting likely spatial relations between objects markedly improves the accuracy of both annotation and placing [12]. Li and Fei-Fei show that event recognition is improved by explicit inference on a generative model representing the scene in which the event occurs and also the objects in the image [13]. Using a different generative model, Li and Fei-Fei demonstrate that relations improve object labels, scene labels and segmentation [14]. Gupta and Davis show that respecting relations between objects and actions improve recognition of each [15, 16]. Yao and Fei-Fei use the fact that objects and human poses are coupled and show that recognizing one helps the recognition of the other [17]. Relations between words in annotating sentences can reveal image structure. Berg *et al.* show that word features suggest which names in a caption are depicted in the attached picture, and that this improves the accuracy of links between names and faces [18]. Mensink and Verbeek show that complex co-occurrence relations between people improve face labelling, too [19]. Luo, Caputo and Ferrari [20] show benefits of associating faces and poses to names and verbs in predicting "who's doing what" in news articles. Coyne and Sproat describe an auto-illustration system that gives naive users a method to produce rendered images from free text

descriptions (Wordseye; [21];`http://www.wordseye.com`).

There are few attempts to generate sentences from visual data. Gupta *et al.* generate sentences narrating a sports event in video using a compositional model based around AND-OR graphs [22]. The relatively stylised structure of the events helps both in sentence generation and in evaluation, because it is straightforward to tell which sentence is right. Yao *et al.* show some examples of both temporal narrative sentences (i.e. this happened, then that) and scene description sentences generated from visual data, but there is no evaluation [23]. These methods generate a direct representation of what is happening in a scene, and then decode it into a sentence.

An alternative, which we espouse, is to build a scoring procedure that evaluates the similarity between a sentence and an image. This approach is attractive, because it is symmetric: given an image (resp. sentence), one can search for the best sentence (resp. image) in a large set. This means that one can do both illustration and annotation with one method. Another attraction is the method does not need a strong syntactic model, which is represented by the prior on sentences. Our scoring procedure is built around an intermediate representation, which we call the **meaning** of the image (resp. sentence). In effect, image and sentence are each mapped to this intermediate space, and the results are compared; similar meanings result in a high score. The advantage of doing so is that each of these maps can be adjusted discriminatively. While the meaning space could be abstract, in our implementation we use a direct representation of simple sentences as a meaning space. This allows us to exploit distributional semantics ideas to deal with out of vocabulary words. For example, we have no detector for "cattle"; but we can link sentences containing this word to images, because distributional semantics tells us that a "cattle" is similar to "sheep" and "cow", etc. (Figure 2.6)

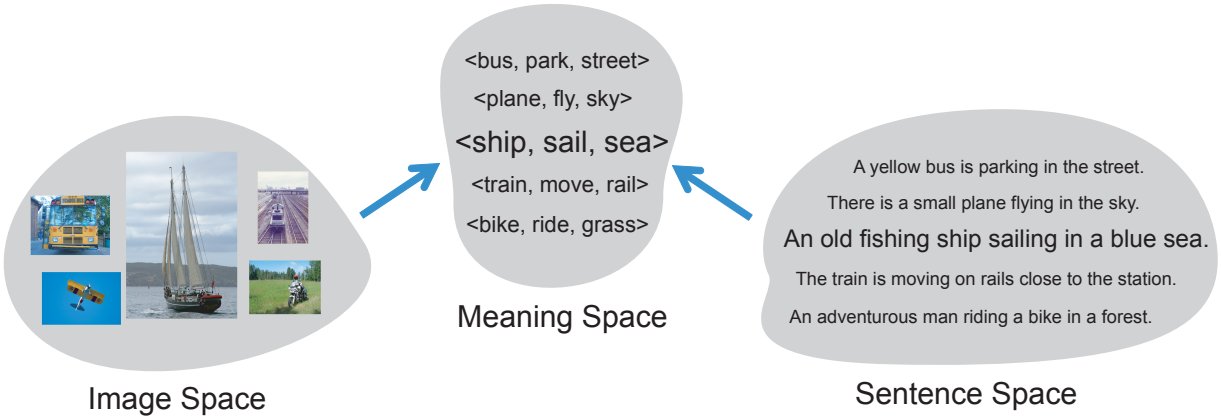


Figure 2.1: There is an intermediate space of meaning which has different projections to the space of images and sentences. Once we learn the projections we can generate sentences for images and find images best described by a given sentence.

2.2 Approach

Our model assumes that there is a space of *Meanings* that comes between the space of *Sentences* and the space of *Images*. We evaluate the similarity between a sentence and an image by (a) mapping each to the meaning space then (b) comparing the results. Figure 2.1 depicts the intermediate space of meanings. We will learn the mapping from images (resp. sentences) to meaning discriminatively from pairs of images (resp. sentences) and assigned meaning representations.

2.2.1 Mapping Image to Meaning

Our representation of meaning in this work a triplet of $\langle object, action, scene \rangle$ (Later, more sophisticated representations were used by other researchers). This triplet provides a holistic idea about what the image (resp. sentence) is about and what is most important. For the image, this is the part that people would talk about first; for the sentence, this is the structure that should be preserved in the tightest summary. For each slot in the triplet, there is a discrete set of possible values. Choosing among them will result in a triplet. The mapping from images to meaning is reduced to learning to predict triplet for images. The problem of predicting a triplet from an image involves solving a (small) multi-label Markov random field. Each slot in the meaning representation can take a

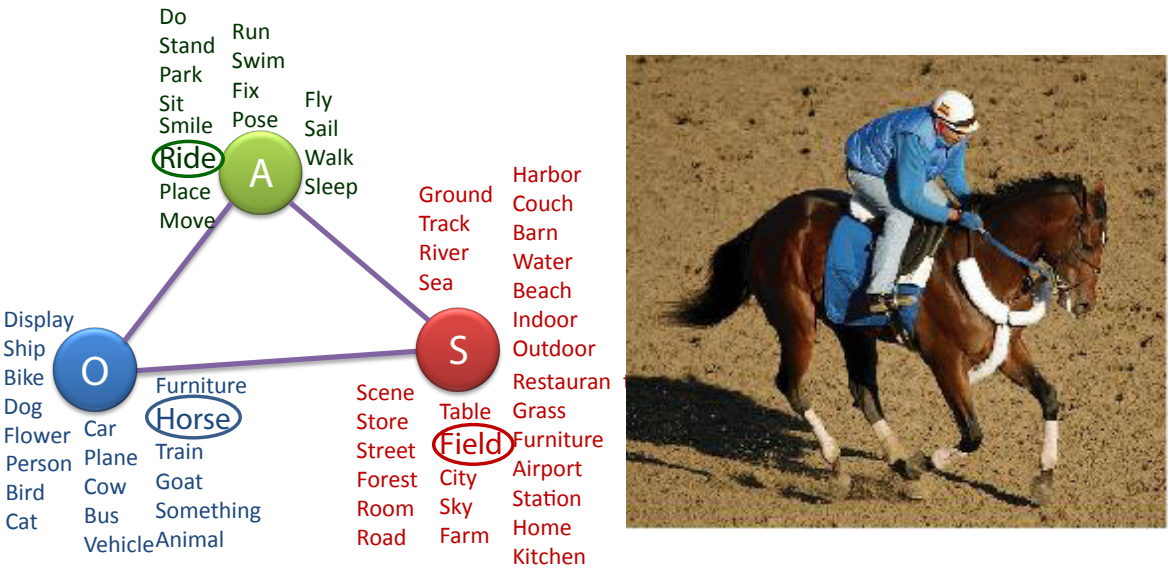


Figure 2.2: We represent the space of the meanings by triplets of $\langle object, action, scene \rangle$. This is an MRF. Node potentials are computed by linear combination of scores from several detectors and classifiers. Edge potentials are estimated by frequencies. We have a reasonably sized state space for each of the nodes. The possible values for each nodes are written on the image. "O" stands for the node for the object, "A" for the action, and "S" for scene. Learning involves setting the weights on the node and edge potentials and inference is finding the best triplets given the potentials.

value from a set of discrete values. Figure 2.2 depicts the representation of the meaning space and the corresponding MRF. There is a node for objects which can take a value from a possible set of 23 nouns, a node for actions with 16 different values, and a node to scenes that can select each of 29 different values. The edges correspond to the binary relationships between nodes. Having provided the potentials of the MRF, we use a greedy method to do inference. Inference involves finding the best selection of the discrete sets of values given the unary and binary potentials.

We learn to predict triplets for images discriminatively. This requires having a dataset of images labeled with their meaning triplets. The potentials are computed as linear combinations of feature functions. This casts the problem of learning as searching for the best set of weights on the linear combination of feature functions so that the ground truth triplets score higher than any other triplet. Inference involves finding $argmax_y w^T \Phi(x, y)$ where Φ is the potential function, y is the triplet label, and w are the learned weights.

2.2.2 Image Potentials

We need informative features to drive the mapping from the image space to the meaning space.

Node Potentials:

To provide information about the nodes on the MRF we first need to construct image features. Our *image features* consist of:

Felzenszwalb *et al.* detector responses: We use Felzenszwalb detectors [24] to predict confidence scores on all the images. We set the threshold such that all of the classes get predicted, at least once in each image. We then consider the max confidence of the detections for each category, the location of the center of the detected bounding box, the aspect ratio of the bounding box, and it's scale.

Hoiem *et al.* classification responses: We use the classification scores of Hoiem et. al [25] for the PASCAL classification tasks. These classifiers are based on geometry, HOG features, and detection responses.

Gist-based scene classification responses: We encode global information of images using gist [26]. Our features for scenes are the confidences of our Adaboost style classifier for scenes.

First we build node features by fitting a discriminative classifier (a linear SVM) to predict each of the nodes independently on the image features. Although the classifiers are being learned independently, they are well aware of other objects and scene information. We call these estimates *node features*. This is a number-of-nodes-dimensional vector and each element in this vector provides a score for a node given the image. This can be a node potential for object, action, and scene nodes. We expect similar images to have similar meanings, and so we obtain a set of features by matching our test image to training images. We combine these features into various other node potentials as below:

- by matching image features, we obtain the k-nearest neighbours in the training set to the test image, then compute the average of the node features over those neighbours, *computed from the image side*. By doing so, we have a representation of what the node features are for similar images.

- by matching image features, we obtain the k-nearest neighbours in the training set to the test image, then compute the average of the node features over those neighbours, *computed from the sentence side*. By doing so, we have a representation of what the sentence representation does for images that look like our image.

- by matching those node features derived from classifiers and detectors (above), we obtain the k-nearest neighbours in the training set to the test image, then compute the average of the node features over those neighbours, *computed from the image side*. By doing so, we have a representation of what the node features are for images that produce similar classifier and detector outputs.

- by matching those node features derived from classifiers and detectors (above), we obtain the k-nearest neighbours in the training set to the test image, then compute the average of the node features over those neighbours, *computed from the sentence side*. By doing so, we have a representation of what the sentence representation does for images that produce similar classifier and detector outputs.

Edge Potentials:

Introducing a parameter for each edge results in unmanageable number of parameters. In addition, estimates of the parameters for the majority of edges would be noisy. There are serious smoothing issues. We adopt an approach similar to Good Turing smoothing methods to a) control the number of parameters b) do smoothing. We have multiple estimates for the edges potentials which can provide more accurate estimates if used together. We form the linear combinations of these poten-

tials. Therefore, in learning we are interested in finding weights of the linear combination of the initial estimates so that the final linearly combined potentials provide values on the MRF so that the ground truth triplet is the highest scored triplet for all examples. This way we limit the number of parameters to the number of initial estimates.

We have four different estimates for edges. Our final score on the edges take the form of a linear combination of these estimates. Our four estimates for edges from node A to node B are:

- The normalized frequency of the word A in our corpus, $f(A)$.
- The normalized frequency of the word B in our corpus, $f(B)$.
- The normalized frequency of (A and B) at the same time, $f(A, b)$.
- $\frac{f(A,B)}{f(A)f(B)}$.

2.2.3 Sentence Potentials

We need a representation of the sentences. We represent a sentence by computing the similarity between the sentence and our triplets. For that we need to have a notion of similarity for objects, scenes and actions in text.

We used the Curran & Clark parser [27] to generate a dependency parse for each sentence. We extracted the subject, direct object, and any nmod dependencis involving a noun and a verb. These dependencies were used to generate the (object, action) pairs for the sentences. In order to extract the scene information from the sentences, we extracted the head nouns of the prepositional phrases (except for the prepositions "of" and "with"), and the head nouns of the phrase "X in the background".

Lin Similarity Measure for Objects and Scenes

We use the Lin similarity measure [28] to determine the semantic distance between two words. The Lin similarity measure uses WordNet synsets as the possible meanings of each words. The noun synsets are arranged in a heirarchy based on hypernym (is-a) and hyponym (instance-of) relations. Each synset is defined as having an information content based on how frequently the synset or a hyponym of the synset occurs in a corpus (in the case, SemCor). The similarity of two synsets is defined as twice the information content of the least common ancestor of the synsets divided by the sum of the information content of the two synsets. Similar synsets will have a LCA that covers the two synsets, and very little else. When we compared two nouns, we considered all pairs of a filtered list of synsets for each noun, and used the most similar synsets. We filtered the list of synsets for each noun by limiting it to the first four synsets that were at least 10% as frequent as the most common synset of that noun. We also required the synsets to be physical entities.

Action Co-occurrence Score

We generated a second image caption data set consisting of roughly 8,000 images pulled from six Flickr groups. For all pairs of verbs, we used the likelihood ratio to determine if the two verbs co-occurring in the different captions of the same image was significant. We then used the likelihood ratio as the similarity score for the positively correlated verb pairs, and the negative of the likelihood ratio as the similarity score for the negatively correlated verb pairs. Typically, we found that this procedure discovered verbs that were either describing the same action or describing two actions that commonly co-occurred.

Node Potentials:

We now can provide a similarity measure between sentences and objects, actions, and scenes using scores explained above. Below we explain our estimates of sentence node potentials.

- First we compute the similarity of each object, scene, and action extracted from each sentence. This gives us the the first estimates for the potentials over the nodes. We call this the *sentence node feature*.

- For each sentence, we also compute the average of sentence node features for other four sentences describing the same images in the train set.

- We compute the average of k nearest neighbors in the sentence node features space for a given sentence. We consider this as our third estimate for nodes.

- We also compute the average of the image node features for images corresponding to the nearest neighbors in the item above.

- The average of the sentence node features of reference sentences for the nearest neighbors in the item 3 is considered as our fifth estimate for nodes.

- We also include the sentence node feature for the reference sentence.

Edge Potentials:

The edge estimates for sentences are identical to to edge estimates for the images explained in previous section.

2.2.4 Learning

There are two mappings that need to be learned. The map from the image space to the meaning space uses the image potentials and the map from the sentence space to the meaning space uses the sentence potentials. Learning the mapping from images to meaning involves finding the weights on the linear combinations of our image potentials on nodes and edges so that the ground truth triplets score highest among all other triplets for all examples. This is a structure learning problem [29] which takes the form of

$$\min_{w} \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i \in examples} \xi_i \tag{2.1}$$

subject to

$$w\Phi(x_i, y_i) + \xi_i \geq \max_{y \in \text{meaning space}} w\Phi(x_i, y) + L(y_i, y)\ \forall i \in \text{examples}$$

$$\xi_i \geq 0\ \forall i \in \text{examples}$$

where λ is the tradeoff factor between the regularization and slack variables ξ, Φ is our feature functions, x_i corresponds to our i^{th} image, and y_i is our structured label for the i^{th} image. We use the stochastic subgradient descent method [30] to solve this minimization.

2.3 Evaluation

We emphasize quantitative evaluation in our work. Our vocabulary of meaning is significantly larger than the equivalent in [13, 14]. Evaluation requires innovation both in datasets and in measurement, described below.

2.3.1 Dataset

We need a dataset with images and corresponding sentences and also labels for our representations of the meaning space. No such dataset exists. We build our own dataset of images and sentences around the PASCAL 2008 images. This means we can use and compare to state of the art models and image annotations in PASCAL dataset.

PASCAL Sentence data set

To generate the sentences, we started with the 2008 PASCAL development kit. We randomly selected 50 images belonging to each of the 20 categories. Once we had a set of 1000 images, we used Amazon's Mechanical Turk to generate five captions for each image. We required the annotators to be based in the US, and that they pass a qualification exam testing their ability to identify spelling errors, grammatical errors, and descriptive captions. More details about the methods of collection can be found in [31]. Our dataset has 5 sentences for each image of the thousand images resulting in 5000 sentences. We also manually add labels for triplets of $\langle objects, actions, scenes \rangle$ for each images. These triplets label the main object in the image, the main action, and the main place. There are 173 different triplets in our train set and 123 in test set. There are 80 triplets in the test set that appeared in the train set. The dataset is available at `http://vision.cs.uiuc.edu/pascal-sentences/`.

2.3.2 Inference

Our model is learned to maximize the sum of the scores along the path identified by a triplet. In inference we search for the triplet which gives us the best additive score, $argmax_y w^T \Phi(x_i, y)$. These models prefer triplets with combination of strong and poor responses over all mediocre responses. We conjecture that a multiplicative inference model would result in better predictions as the multiplicative model prefers all the responses to be reasonably good. Our multiplicative inference has the form of $argmax_y \prod w^T \Phi(x_i, y)$. We select the best triplet given the potentials on the nodes and edges greedily by relaxing an edge and solving for the best path and re-scoring the results using the relaxed edge.

2.3.3 Matching

Once we predict triplets for images and sentences we can score a match between an image and a sentence. If an image and a sentence predict very similar triplets, they should be projections of nearby points in the meaning space, and so they should have a high matching score. A natural score of the similarity of sentence triplets and image triples is the sum of ranks of sentence meaning and image meaning; the pair with smallest value of this sum is both strongly predicted by the image and strongly predicted by the sentence. However, this score is likely to be noisy, and is difficult to compute, because we must touch all pairs of meanings. We use a good, noise resistant approximation. To obtain the score, we:

- obtain the top k ranking triplets derived from sentences and compute the rank of each as an image triplet
- obtain the top k ranking triplets derived from images and compute the rank of each as a sentence triplet
- sum the sum of ranks for each of these sets, weighted by in the inverse rank of the triplet, so as to emphasize triplets that score strongly.

2.3.4 Out of Vocabulary Extension

We generate sentences by searching a pool of sentences for one that has a good match score to the image. We cannot learn a detector/classifier for each object/action/scene that exists. This means we need to score the similarity between the image and sentences that contain unfamiliar words. We propose using text information to attack this problem. For each unknown object we can produce a score of the similarity of that object with all of the objects in our vocabulary using distributional semantics methods explained in section 2.2.3 . We do the same thing for verbs and scenes as well. These similarity measures work as a crude guide to our model. For example, in

Figure 2.6, we don't have a detector for "Volkswagen", "herd", "woman", and "cattle" but we can recognize them. our similarity measures provides a similarity distributions over things we know. This similarity distribution helps us to recognize objects, actions, and scenes for which we have no detector/classifier using objects/actions/scenes we know.

2.3.5 Experimental settings

We divide our 1000 images to 600 training images and 400 testing images. We use 15 nearest neighbors in building potentials for images and sentences. For matching we use 50 closest triplets.

2.3.6 Mapping to the Meaning Space

Table 2.1 compares the results of mapping the images to the meaning space, predicting triplets for images. To do that, we need a measure of comparisons between pairs of triplets, the one that we predict and the ground truth triplets. One way of doing this is by simple comparisons of triplets. A prediction is correct if all three elements agree and wrong otherwise. We could also measure if any of the elements in the triplet match. Each score is insensitive to important aspects of loss. For example, predicting $\langle cat, sit, mat\rangle$ when ground truth is $\langle dog, sit, ground\rangle$ is not as bad as predicting $\langle bike, ride, street\rangle$. This implies that the penalty for confusing cats with dogs should be smaller than that for confusing cats with bikes. The same argument holds for actions and scenes as well. We also need our measure to take into account the amount of information a prediction conveys. For example, predicting $\langle object, do, scene\rangle$ is less favorable than $\langle cat, sit, mat\rangle$.

Tree-F1 measure:

Tree-F1 measure: We need a measure that reflects two important interacting components, accuracy and specificity. We believe the right way to score error is to use taxonomy trees. We have taxonomy trees for objects, actions, and scenes and we can use them to measure the accuracy, relevance, and

specificity of predictions. We introduce a novel measure, Tree-F1, which reflects how accurate and specific the prediction is. Given a taxonomy tree for, say, objects objects, we represent each prediction by the path from the root of the taxonomy tree to the predicted node. For example, if the prediction is cat we represent it as $Objects \Rightarrow animal \Rightarrow cat$. We can then report the standard F1 measure using the precision and recall. Precision is defined as the total number of edges on the path that matches the edges on the ground truth path divided by the total number of edges on the ground truth path and recall as the total number of edges on the predicted path which is in the ground truth path divided by the total number of edges in the path. For example, the measure for predicting dog when the ground truth is cat is 0.5 where the precision is 0.5 and recall is 0.5, the measure for predicting animal when the ground truth is cat is 0.66, and it is 0 for predicting bike when the ground truth is cat. The same procedure is applied to actions and scenes. The Tree-F1 measure for a triple is the mean of the three measures for objects, actions, and scenes. Table 2.1 shows Tree-F1 measures for several different experimental settings.

BLUE Measure:

Similar to Machine translation approaches where reports of accuracy involves scores for the correctness of the translation and the correctness of the generated translation in terms of language and logic, we also consider another measure to check if the triplet we generate is logically valid or not. Analogous to the BLEU score in machine translation literature we introduce the "BLUE" score which measures this. For example, $\langle bottle, walk, street \rangle$ is not valid. For that, we check if the triplet ever appeared in our corpus or not. Table 2.1 shows these scores for the triplets predicted by several different experimental settings.

	Obj	No Edge	FW(A)	SL(A)	FW(M)	SL(M)
Mean Tree-F1 for first 5	0.44	**0.52**	0.38	0.45	0.47	0.51
Mean BLUE for first 5	0.24	0.27	0.16	0.58	**0.76**	0.74
Mean Tree-F1 for first 5 objects	**0.59**	0.58	0.36	0.53	0.55	0.57
Mean Tree-F1 for first 5 actions	0.27	**0.52**	0.50	0.37	0.42	0.47
Mean Tree-F1 for first 5 scenes	0.28	0.48	0.28	0.44	0.46	**0.48**

Table 2.1: Evaluation of mapping from the image space to the meaning space. "Obj" means when we only consider the potentials on the object node and use uniform potentials for other nodes and edges. "No Edge" means assuming a uniform potential over edges. "FW(A)" stands for fixed weights with additive inference model. This is the case where we use all the potentials but we don't learn any weights for them. "SL(A)" means using structure learning with additive inference model. "FW(M)" is similar to "FW(A)" with the exception that the inference model is multiplicative instead of additive. "SL(M)" is the structure learning with multiplicative inference.

2.4 Results

To evaluate our method we provide qualitative and quantitative results. There are two stages in our model. First we show the ability of our method to map from the image space to the meaning space. We then evaluate our results on predicting sentences for images, annotation. We also show qualitative results for finding images for sentences, illustration.

2.4.1 Mapping Images to Meanings

Table 2.1 compares several different experimental settings in terms of two measures explained above, Tree-F1 and BLUE. Each column in Table 2.1 corresponds to an experimental setting. We report average Tree-F1 and average BLUE measures for five top triplets for all images. We also breakdown the Tree-F1 to objects, actions, and scenes in bottom three rows of the table.

2.4.2 Annotation: Generating Sentences from Images

Figure 2.3 shows top 5 predicted triplets and top 5 generated sentences for example images in our test set. Quantitative evaluation of generated sentence is very challenging. We trained 2 individuals to annotate generated sentences. We ask them to annotate each generated sentence by either 1, 2,

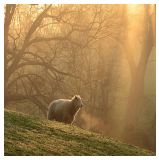
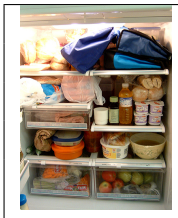
	(pet, sleep, ground) (dog, sleep, ground) (animal, sleep, ground) (animal, stand, ground) (goat, stand, ground)	see something unexpected. Cow in the grassfield. Beautiful scenery surrounds a fluffly sheep. Dog hearding sheep in open terrain. Cattle feeding at a trough.
	(furniture, place, furniture) (furniture, place, room) (furniture, place, home) (bottle, place, table) (display, place, table)	Refrigerator almost empty. Foods and utensils. Eatables in the refrigerator. The inside of a refrigerator apples, cottage cheese, tupperwares and lunch bags. Squash apenny white store with a hand statue, picnic tables in front of the building.
	(transportation, move, track) (bike, ride, track) (transportation, move, road) (pet, sleep, ground) (bike, ride, road)	A man stands next to a train on a cloudy day A backpacker stands beside a green train This is a picture of a man standing next to a green train There are two men standing on a rocky beach, smiling at the camera. This is a person laying down in the grass next to their bike in front of a strange white building.
	(display, place, table) (furniture, place, furniture) (furniture, place, furniture) (bottle, place, table) (furniture, place, home)	This is a lot of technology. Somebody's screensaver of a pumpkin A black laptop is connected to a black Dell monitor This is a dual monitor setup Old school Computer monitor with way to many stickers on it

Figure 2.3: Generating sentences for images: We show top five predicted triplets in the middle column and top five predicted sentences in the right column.

or 3. 1 means that the sentence is quite accurate with possible little mistakes about details in the sentence. 2 implies that the sentence have a rough idea about the image but it's not very accurate and 3 means that the sentence is not even remotely close to the image. We generate 10 sentences for each image. The total average of the scores given by these individuals is 2.33. The average number of sentences with score one per image is 1.48. The average number of sentences with score 2 per image is 3.8. 208 of 400 images have at least one sentence with score 1. 354 sentences out of 400 images have at least one sentence with score 2.

A two girls in the store.

Yellow train on the tracks.

A small herd of animals with a calf in the grass.

A horse being ridden within a fenced area.

Figure 2.4: Finding images for sentences: Once the matching in the meaning space is established we can generate sentences for images (annotation) and also find images that can be best describe by a sentence. In this picture we show four sentences with four 144 highest ranked images. We provide a list of 10 highest score images for each sentence for the test set in the supplementary material.

	A male and female giving pose for camera. A peaceful garden The food is ready on table.
	The two girls read to drive big bullet. Man with a goatee beard kneeling in front of a garden fence. Lone bicyclist sitting on a bench at a snowy beach.
	Black goat in a cage Horse behind a fence Wooly sheep standing next to a fence on a sunny day.

Figure 2.5: Examples of failures in generating sentences for images.

2.4.3 Illustration: Finding images best described by sentences

Not only our model can provide sentences that describe an image, but it also can find images which are best described by a given sentence. Once the connections to the meaning space is established, one could go in both directions, from images to sentences or the other way around. Figure 2.4 shows examples of finding images for sentences. For more qualitative results please see the supplementary material.

2.4.4 Out of Vocabulary Extension

Figure 2.6 depicts examples of the cases where we could successfully recognize objects/actions for which we have no detector/classifier. This is very interesting as the intermediate meaning space allows us to benefit from distributional semantics. This means that we can learn to recognize

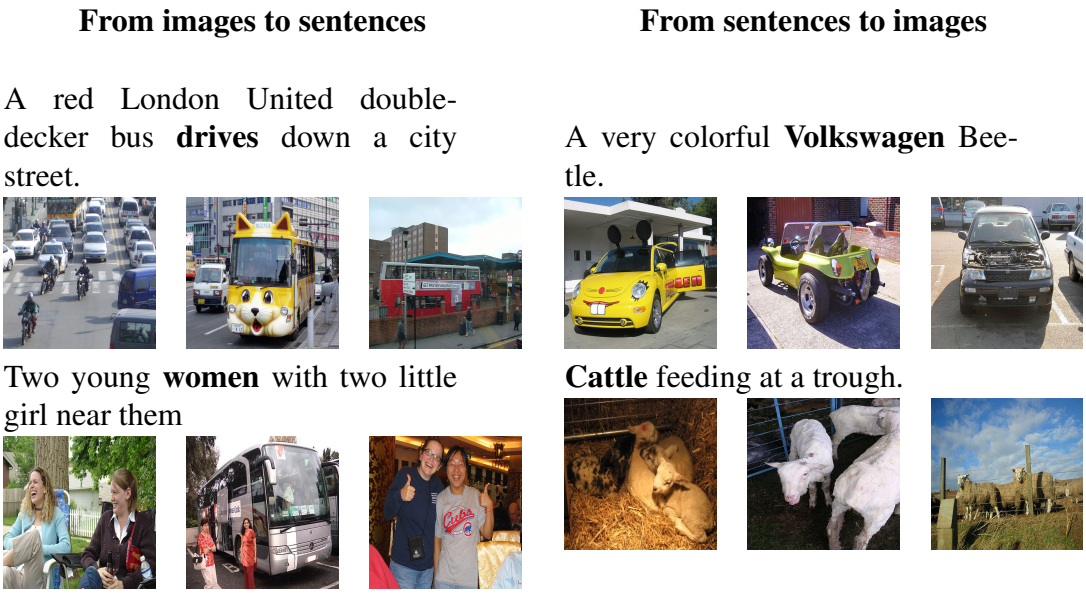


Figure 2.6: Out of vocabulary extension: We don't have detectors for "drives", "women", "Volkswagen", and "Cattle". Despite this fact, we could recognize these objects/actions. Distributional semantics provide us with the ability to model unknown objects/actions/categories with their similarities to known categories. Here we show examples of sentences and images when we could recognize these unknowns for both generating sentences from images and finding images for sentences.

unknown objects/actions/scenes by looking at the patterns of responses from other similar known detector/classifiers.

2.5 Follow-up Work

Sentences are rich, compact and subtle representations of information. Even so, we can predict good sentences for images that people like. The intermediate meaning representation is one key component in our model as it allows benefiting from distributional semantics. Since the publication of this work, this topic has grown and flourished. Follow-up work has improved all aspects of our pipeline: Improved object detection, improved image description, improved language model, and improved datasets.

In summary, four aspects have helped improve sentence generation since 2010. (1) Larger sentence datasets, (2) Improved language model, (3) Improved object detectors, (4) Improved

modeling of image description. In the next few subsections we will discuss the follow-up work.

2.5.1 Larger Datasets

Kulkarni et. al [32] uses an expanded Conditional Random Field to cover more content from the image. This work is able to produce richer descriptions that include multiple sentences and adjectives. Ordonez et. al [33] expand the size of their dataset 100-fold to to about 1 Million images. The large size of their dataset allows for better matches in the dataset and subsequently improved descriptions.

Mitchell et. al [35] use a dataset of 700,000 Flickr images to learn word co-occurrence statistics. They decode object detections to filter out unreliable detections. They form a tree that describes what their system sees. Finally they generate sentences using these trees.

2.5.2 Improved Language Model

Today sentence datasets have grown two to three orders of magnitude. The improvement by larger datasets is expected, however larger dataset have posed new computational and algorithmic challenges that researchers have studied. A major improvement comes from language models that can produce novel sentences [40] [41]. Handling a language model needs several technical considerations and the more complex language models come with more susceptibility to error. A significant portion of contribution on this topic has been various language models to generate sentences with diverse characteristics.

Yang et. al [34] expands our algorithm by using a language model trained from a corpus of sentences. This language model is able to construct novel sentences using a Hidden Markov Model.

Hodosh et. al [36] study sentence-based image annotation as the task of ranking a given pool of captions. They perform an in-depth comparison of human and automatic evaluation metrics

for this task, and propose strategies for collecting human judgements cheaply and on a very large scale. They suggest that the evaluation of ranking-based image description systems could be done automatically. Young et. al [37] propose to use the visual denotations of linguistic expressions (i.e. the set of images they describe) to define novel denotational similarity metrics. They show that visual detonations are viable descriptions and can be as good as distributional similarities for two tasks. The novelty in this work is that image-sentence datasets can be analysed using multiple different data structures.

Das et. al [38] apply similar techniques to videos. They extract several keywords from a video segment and use them to generate sentences. Their system is able to describe a video using multiple sentences. These sentences can tell a story of what happens in the video.

2.5.3 Improved Object Detection

To a human observer, the quality of object detection could be more apparent than the quality of the language. Older sentence generation methods focused on how to handle noise in the object detection. Later methods, in contrast [41], do not need to focus on noise handling. Perhaps the largest effect on the quality of sentence generation comes from the improved quality of object detection.

A significant portion of the contribution before 2014 was how to handle noise in the input detectors. Since 2014 several groups have started using Convolutional Neural Networks (CNN) because of their higher accuracy. The benefit of CNNs is that raw image descriptions are more reliable. This is crucial because sentence generation involves multiple steps, each with some inherent error. CNN based approaches worry less about cleaning up detection data. Instead, they focus on improving language model and how image descriptions are used.

2.5.4 Improved Image Modeling

Karpathy et. al [39] model images in a more careful way. They produce descriptions for local patches of images using CNNs. They then learn to produce sentences by observing spatial relations between local patches.

Socher et. al [40] use an intermediate space between images and sentences. However, their intermediate space is richer than the one we first proposed in our work. They use CNNs to map images into the intermediate space. They also use a recursive neural network to map sentences into the intermediate space. After they map both images and sentences into the intermediate space they can perform several tasks including describing images with sentences. Vinyals et. al [41] take a systematic approach to generating sentences for images. They first map images into a description using CNNs. Then, they use a language model to generate sentences. They train the whole model to maximize the likelihood of the target description sentence given the training image.

2.6 Discussion and Future Work

We believe there is still room to improve upon the state-of-the-art. improved language models with larger and more diverse datasets would improve the quality and diversity of sentences.

Perhaps one area of research in future would be question answering and entailment. For example, a system that can answer a human input question on an image. Such a system would need to understand both the image and the question and be able to reason about their relation. Another example would be a system trained for a pre-specified purpose such as describing scenes specifically for the blind. In a more general case, a system could be presented with a a set of images, some textual description and a question. Then, the system would have to reason about all the input and answer the question.

Chapter 3

Recognition using Visual Phrases

3.1 Introduction

There has been significant progress in building detection algorithms that can find instances of designated objects in images. This is a challenging task partly due to intraclass variations; objects of the same category seems very different. Pose variation due to actions, interaction with other objects, spatial relations that cause partial occlusions, are among important reasons of intraclass variations. The conventional approach to encode these kinds of variations is to use a mixture of multiple models for each category. These mixture models treat all variations equally and learn a model for each group of variations. We show that not all intraclass variations are equally hard to encode, suggesting an asymmetric model that treats different variations differently.

What makes some variations easier to encode? We show that some actions, interactions, or spatial relations impose a very characteristic appearance that makes detecting the object of interest significantly easier. For example, interactions between objects imposes rigid constraints on the appearance of objects. How should one detect complex visual composites, for example "a person riding a horse"? Conventional wisdom suggests detecting components like "person" and "horse" independently, and then describing the relation. This approach is motivated by the very large number of composites that can be built by very few basic atoms. Also, there will be very few training examples for most composites due to the increase in specifications.

The main weakness of this argument is that the appearance of the objects may profoundly change when they participate in relations. For example, people riding horses take relatively few

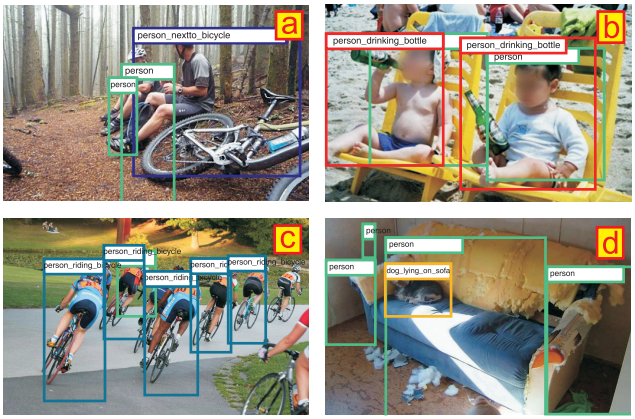


Figure 3.1: Detecting visual phrases is often significantly more accurate than detecting participating objects. In image "a", the bicycle detector and the person detector do not have accurate responses whereas our "person next to bicycle" detector correctly finds the visual phrase. In image "b", the bottle detector does not produce any sensible detection while our "person drinking from bottle" detector accurately finds instances of the visual phrase. The faces of the children are blurred here due to privacy concerns. In image "c", the person detector could only find one instance of a person while our "person riding bicycle" detector finds 5 instances correctly. In image "d", neither the dog detector nor the sofa detector is producing reliable responses but our "dog lying on sofa" detector finds the visual phrase correctly. We believe that detecting visual phrases are often much easier than the participating objects as visual phrases exhibit less visual complexity. See Figure 3.4, and Table 3.1 for quantitative evaluations.

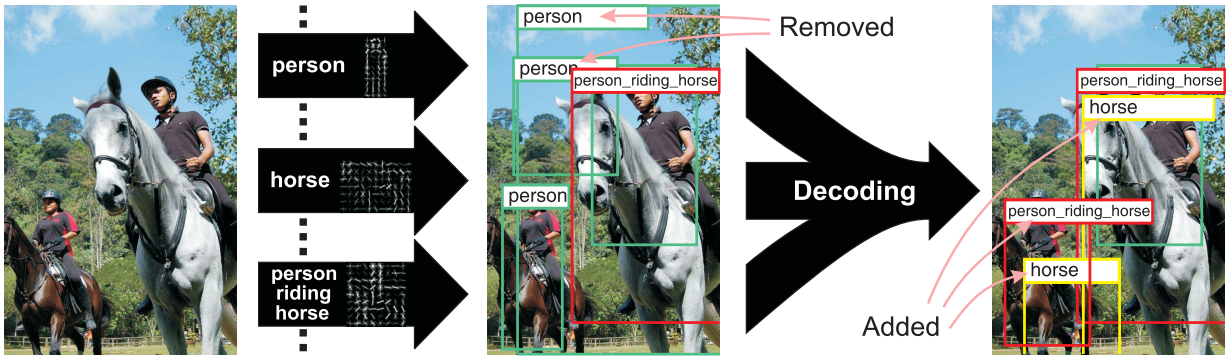


Figure 3.2: We use visual phrase and object models to make independent predictions. We then combine the predictions by a decoding algorithm that takes all detection responses and decides on the final outcome. Note that a) Visual phrase recognition works better than recognizing the participating objects. For example, the horse detector does not produce reliable predictions about horses in this picture while the "person riding horse" detector finds one instance; b) Our decoding then successfully adds two examples of horses and removes two wrong predictions of people by looking at other detections in the vicinity.

postures, as do horses with people on their back. Relations may also create important occlusion regularities. For instance, one leg of the rider is often occluded by the horse. As a result, visual composites might be much easier to detect than their participant components. The same argument holds for some actions. For example, dogs typically take a very characteristic posture when they jump to catch a Frisbee. A general dog detector will have a hard time detecting those instances of dogs because of their specific posture. This characteristic posture that causes serious issues for a generic dog detector is the main reason that makes detecting "dog jumping" much easier. Spatial relationships between objects are also the same. For example, when people stand next to a bike they appear in a specific location with respect to the bike, which, makes detecting that person much easier.

One extreme example is a scene (e.g. kitchen). There are quite good "kitchen" classifiers, but none proceeds by finding "toaster", "coffee-pot", and "kettle", then fusing. Prior to our CVPR'11 version of this work [2] composite intermediates between objects and scenes were not studied in the literature. We introduce such intermediate composites, which we call "***visual phrases***". Visual phrases correspond to chunks of meaning bigger than objects and smaller than scenes. We show that the reduction in the visual complexity exhibited by visual phrases is often so great that very accurate detectors can be trained with little training data. For example, our "person riding horse" detector works much better than "person" and "horse" detectors while using less training data (see Figure 3.4 for experimental data). Figure 3.1 shows examples of the cases where best object detectors miss objects while the visual phrase detectors correctly localize visual phrases. We believe that the current choice of categories as basic atoms of recognition is arbitrary. We argue that these basic atoms should be chosen by performance criteria. Opportunism is the key to this principle.

Adding visual phrases to the vocabulary of recognition raises interesting problems. A strong "person riding horse" detection should provide strong cues to a "person" and a "horse" predictions and vice versa. It also should discourage predictions of irrelevant categories; for example an

Figure 3.3: The phrasal recognition dataset consists of 17 phrases and 8 objects. There are 2769 images in this dataset and on average 120 images per category. This figure shows 6 example of 7 different visual phrases in our dataset. Rows correspond to visual phrases: dog jumping; horse and rider jumping; person drinking from bottle; person jumping; person lying on beach; person lying on sofa; person next to bicycle.

airplane prediction is highly unlikely to significantly overlap with person riding horse, person, and horse. Detecting multiple overlapping objects require a reasoning module that takes all the predictions as input and decides on the final outcome. We call this module a decoder. For example in Figure 3.2 the existence of a strong person riding horse detection has convinced our decoding algorithm to encourage a horse prediction.

Phrasal recognition is analogous to machine translation problems where the alignment has to be established between phrases and areas of images. One might think of our system as having a phrase table with entities like “person”, “horse”, and “person riding horse”. The ultimate goal

is to look at all phrases and find the longest phrase that matches. This procedure is often called decoding in machine translation. Our decoder has to take into account that some of the detectors should overlap and when they overlap it has to decide which of the overlapping detectors are worth reporting.

We show the benefits of opportunistically selecting basic atoms of recognition and the significant gain in directly detecting visual phrases. Our contributions are:

1. Introducing a novel dataset for phrasal recognition;
2. Showing that considering visual phrases provides a significant gain over state of the art object detectors coupled with the state of the art methods of modeling interactions;
3. Introducing a decoding algorithm that takes into account specific properties of interacting objects in multiple levels of abstraction;
4. Improving multi-class object recognition using visual phrases;
5. Performing detailed analysis of the effects of specificity in phrasal recognition, reduction in visual complexity, and the separability of negative examples.

3.2 Phrasal Recognition

Our task is to learn appearance models not only for basic level categories but also for richer levels of abstractions, visual phrases. Having learned these appearance models, we show significant gains in considering some visual phrases as a whole instead of detecting the basic atoms and then modeling the interactions, see Figure 3.4 and Table 3.1. We also consider the problem of object recognition in a multi-class framework and model the interactions between categories which includes objects and visual phrases. We show significant boost in multi-class recognition performance using our decoding method along with our visual phrase models comparing to the state of the art basic level models coupled with the state of the art interaction models.

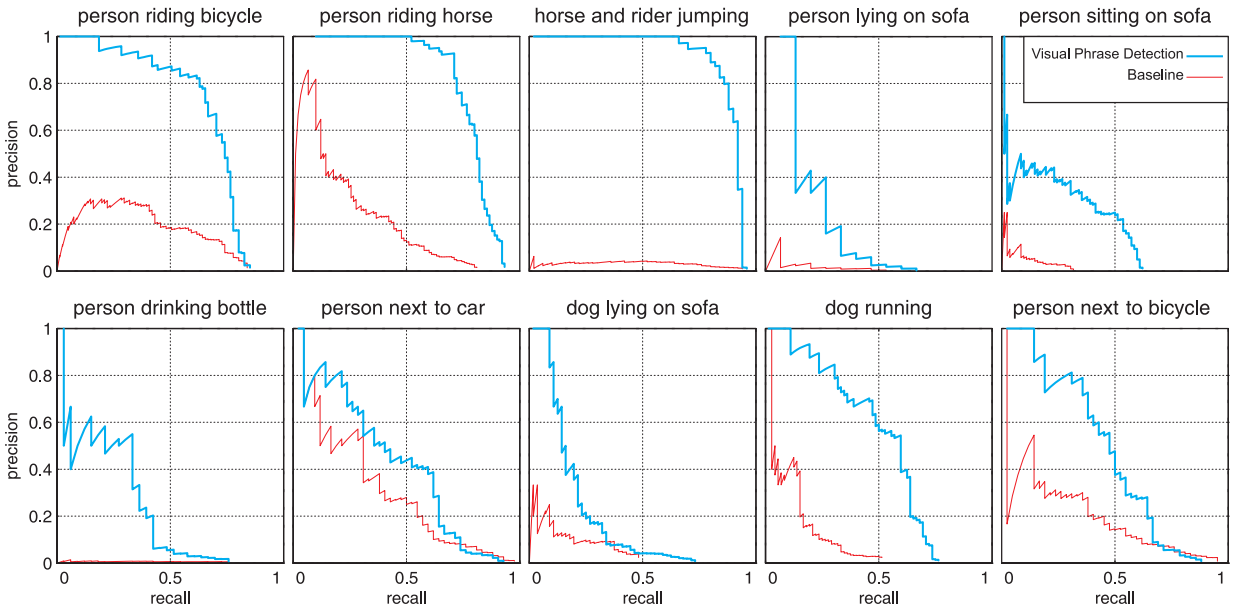


Figure 3.4: Precision-Recall curves for detecting 10 visual phrases in our dataset comparing to the baseline. The comparison to this baseline is biased toward best possible outcome on the test set. Please see section 3.4.1 for more information the baseline. Note the significant gain in detecting visual phrases compared to detecting objects and describing their relations. The gain is astonishing because the phrase detectors are trained using at most 50 positive training examples with default settings while the object detectors are heavily fine tuned and trained using thousands of examples. Further, The baseline is heavily biased toward best possible outcome on the test set. Please see Table 3.1 for detailed AP's for all of the visual phrases in our dataset.

In order to study visual phrase recognition, we need to have a dataset of visual phrases and objects. In the next subsection we introduce a new dataset suitable for phrasal recognition.

3.2.1 Phrasal Recognition Dataset

We first select 8 object classes from Pascal VOC2008 dataset [47] that are suitable for modeling the interactions between objects: person, bike, car, dog, horse, bottle, sofa, and chair. We then add a list of 17 visual phrases using 8 selected object classes. Our visual phrases are formed by either an interaction between objects or activities of single objects. These visual phrases are: person riding horse; person sitting on sofa; person sitting on chair; person lying on sofa; person lying on beach; person riding bicycle; horse and rider jumping; person next to horse; person next to

bicycle; bicycle next to car; person jumping; person next to car; dog lying on sofa; dog running; dog jumping; person running; and person drinking from a bottle. We also add a background class.

We use Bing image search to gather images for the phrases and manually filter out irrelevant images. For basic level categories we used Pascal images. We manually obtain bounding boxes of all the 8 objects along with 17 phrases for all of the images in the dataset. There are 2769 images (822 negative images) in our dataset and on average each class has 120 examples. In total there are 5067 bounding boxes (1796 for visual phrases+3271 for objects) in this dataset. As expected, the number of training examples decreases as the complexity of the phrase increases. However, the collapse in the visual complexity of phrases is so great that one doesn't need to have many training examples to learn visual phrases(see section 3.2.2). This dataset and the phrase models are publicly available at `http://vision.cs.uiuc.edu/phrasal/`. Figure 3.3 shows examples of images in our dataset.

3.2.2 Appearance models

The appearance models for each category, including objects and visual phrases, are learnt using the deformable part models [48]. We learn these models for each of our 17 phrases in our dataset using provided bounding boxes. Available models on the 8 categories from Pascal [48] are used as models for objects in the phrasal recognition dataset. We use these models to evaluate the benefits of phrasal recognition. Many of visual phrase detectors have accurately learned the phrase, Figure 3.4. This is mainly due to the fact that often the appearance of visual phrases has limited variance comparing to the objects in the phrase. For the same reason, the number of necessary training examples for training appearance models for visual phrases can often be very small. Similar to object detectors, some of the visual phrases are hard to train as they have higher variance in the appearance.

3.3 Decoding Multiple Detections

Adding visual phrases to the vocabulary of recognition raises interesting problems. A strong "person riding horse" detection should provide strong cues toward predictions about "person" and "horse". It also should discourage predictions of irrelevant categories; for example an airplane prediction is highly unlikely to significantly overlap with person riding horse, person, and horse. Detecting multiple overlapping objects require a reasoning module that takes all the predictions as input and decides on the final outcome. We call this module a decoder. For example in Figure 3.2 the existence of a strong person riding horse detection has convinced our decoding algorithm to encourage a horse prediction. Decoding takes all detector responses as input and decides on the final outcome. Non-maximum suppression (NMS) is the usual form of decoding. Perfect detectors with excellent tightly tuned models should seldom, if ever, need decoding because there is no ambiguity in what to report. Current detectors are not perfect so decoding is a necessary part of every multiclass object detection method.

One natural decoding strategy, which outperforms NMS, is to model the interaction between objects by having pairwise terms in the scoring function [43]. This approach often yields intractable inferences and one needs to greedily search the space of labels. Pairwise terms are used to model interactions between objects resulting in fiercely intractable combinatorial problems which are hard to approximate.

Our philosophy is that well designed feature representations should make it unnecessary to account for pairwise interactions. To do that, detector responses should be aware of other detectors in a vicinity. We explicitly encode this in our feature representation resulting in very fast, exact inference methods.

Notation: Following the notation of [43], an image is represented as a collection of overlapping bounding boxes which are represented by features x_i. Write $X = \{x_i : i = 1...M\}$ as the representation of an image where M is the total number of bounding boxes for an image. To get

these bounding boxes we run all of the detectors on all of the images. For each bounding box, we know its position, scale, and the confidence of the detector that reported this bounding box. We also assume that there are K different categories and $y_i \in \{0, 1\}$ is the label for each bounding box. $y_i = 1$ means that the i^{th} bounding box should be considered in the final response and $y_i = 0$ is otherwise. $Y = \{y_i : i = 1...M\}$ is the entire label for image X. $c_i \in \{0, 1, ..., K\}$ is the indicator variable showing the category detector that selected the i^{th} bounding box. The score of labeling image X with label Y is defined as $S(X, Y) = \sum_i w_{c_i}^T x_i$ where i is the index to the i^{th} bounding box in image X and w_{c_i} is the set of weights that corresponds to the class of the i^{th} bounding box. We do not consider the pairwise relationships in the scoring function as these relationships are encoded in our feature representation (section 3.3.1).

3.3.1 Representation

We expect our final score for each bounding box to be aware of the results of all other categories nearby. We explicitly encode this in our feature representations. Our representation of an image is based on representations of bounding boxes obtained on each image using all detectors and consists of confidences, the amount of overlap and size ratio of neighboring bounding boxes. To do that we run all of our detectors on each of the images. We consider three spatial relationships: above, below, and overlapping. For each window, for each category, and for each of these spatial bins we consider the confidence of the best scoring window, its overlap, and its size ratio to the represented window. We also add the confidence of the represented window to the features. This means that our representation has $K \times 3 \times 3 + 1$ dimensions.

3.3.2 Inference

We assume bounding boxes are independent given their features. Our feature design makes this assumption reasonable and so our inference is exact. Our inference is

$$
\begin{aligned}
Y^* &= \{y_i^*, i = 1...M\} \\
{y_i}^* &= \arg\max_{y_i} w_{c_i}^T \Phi(X, y_i)
\end{aligned} \tag{3.1}
$$

where i is the index to bounding boxes and w_{c_i} is the corresponding weights for the class of the i^{th} bounding box and $\Phi(X, y_i)$ generates features for that bounding box. This is very simple exact inference as $y_i \in \{0, 1\}$ and y_i's are independent.

3.3.3 Learning

Our model is a form of max margin structure learning. The structured label Y has to be predicted using our decoding model. The objective function takes the form of:

$$
\begin{aligned}
&\min_{w,\xi} \sum_{c\in\{0,...,K\}} \frac{1}{2} \parallel w_c \parallel_2^2 + \lambda \sum_{n}^{N} \xi_n \\
&s.t. \forall n, H_n,\ S(X_n, Y_n) - S(X_n, H_n) \geq \boldsymbol{L}(Y_n, H_n) - \xi_n
\end{aligned} \tag{3.2}
$$

where $n \in \{1, ..., N\}$ is the index to the image and $\boldsymbol{L}$ is the loss between the hypothesis $H_n = \{h_{n,i}, h_{n,i} \in \{0, 1\}, i = 1...M\}$ and the true structured label Y_n, ξ_n is a slack variable, and λ is the tradeoff between the regularization and loss. This max margin formulation requires all of the hypotheses to score lower than the ground truth labels by at least the amount of loss. We model the loss as hamming loss. Eq. 3.2 can be reformulated as

$$\min_{w} \sum_{c\in\{0,\ldots,K\}} \frac{1}{2} \parallel w_c \parallel_2^2 + \tag{3.3}$$

$$\lambda \sum_{n}^{N} \sum_{i}^{M} w_{c_i}^T(\phi(X_n, h_{n,i}^*) - \phi(X_n, y_{n,i})) + \boldsymbol{L}(H_n^*, Y_n)$$

$$\text{s.t. } H_n^* = \arg\max_{H_n} \sum_{i}^{M} w_{c_i}^T \phi(X_n, h_{n,i}) + L(H_n, Y_n) \tag{3.4}$$

Fortunately, in this min-max formulation, our inner maximization is exact and very fast. We solve this optimization problem by subgradient descent method as follows.

We first randomly initialize w_{c_i}'s and solve for H^*'s in the inner maximization problem, Eq 3.4. This is an easy maximization as $h_i \in \{0, 1\}$ and the labels for bounding boxes are independent given their features. We then fix the H^*'s and use the subgradient of the objective function to minimize it. The step size is $1/t$ where t is the number of iterations. Having taken one step, we fix w_{c_i}'s and search for H^* again. We iterate till we converge. The convergence criteria is set by looking at the consecutive improvements on the objective value.

When converged, we use $w_{c_i}^*$ in the inference model (Eq. 3.1) to rescore the bounding boxes accordingly and also infer the final labels.

3.4 Results

To evaluate phrasal recognition and our decoding method we show extensive quantitative results on two tasks: a) single category detection, and b) decoding: multi-category detections. We compare our results to state-of-the-art performance results in both tasks.

Figure 3.5: Phrasal recognition significantly outperforms detection of participating objects and then modeling their interactions. This figure shows examples of visual phrase detections where independent objects couldn't be found using state of the art object models. For example, in image "a", the person detector failed to localize the lady in the red dress while our "person next to bicycle" detector localizes her accurately. In image "b", the person detector fails to localize the baby and our "person drinking from bottle" detector correctly finds this visual phrase.

Figure 3.6: Rows 1 and 2 depicts our results before and after decoding, respectively. The same applies to rows 3 and 4. For example, in image "a", our decoding boosts the confidence of the bicycle classifier and suppresses the confidences of wrong person detections using a reliable "person riding bicycle" detection. In image "c", a confident "dog lying on sofa" detector improves the confidence of the sofa detection and decreases the confidences of wrong person detections. In image "d", the "person sitting on chair" detector increases the confidence of the chair detection. Our decoding shows that visual phrases help object detection and vice versa. In image "b", the confident sofa detection boosts the confidence of "dog lying on sofa" detection.

3.4.1 Single Category Detection

We use deformable part models with default settings [49] to train detectors for our 17 visual phrases. For objects we use the trained models from [48]. These models produce state of the art results in the single object detection task on Pascal dataset. We show significant gain in modeling the visual phrases comparing to separately detecting participating objects and then modeling the relations. Figure 3.4 shows Precision-Recall (PR) curves for some of the visual phrase detectors. We trained these detectors with at most 50 positive examples. Many of the visual phrase detectors produce promising results. To further demonstrate the substantial gain in considering visual phrases, we compare our visual phrase detectors with a baseline that tries to best model interactions between objects.

The baseline takes the confidence responses of participating object detectors as input and tries to best model the interactions between the objects. It is challenging to build a perfect detector that takes into account interactions of objects. We, therefore, build a baseline detector that performs on the *test set* as best as it can. The performance of the baseline can be regarded as an optimistic upper bound on how well one could detect visual phrases by detecting participating objects using the best current detectors. We run detectors for each of the participating objects and consider overlapping responses. There are multiple ways of modeling the interactions between objects: a) We extend the bounding boxes of the overlapping responses of participating objects to estimate the bounding box of the visual phrase. We then compute the average of the confidences of the bounding boxes of the participating objects to estimate a score for the estimated bounding box. We then use this score to produce the PR curves. b) This is similar to "a" but we consider the minimum of the confidences of participating objects rather than their average. c) This is similar to "a" and "b" but we use maximum confidence instead of the average or the minimum. d) We regress the position, scale, and confidence of the final phrase prediction against the positions, scales, and confidences of the participating objects on the *test set*. To produce the best possible outcome, we run all of these procedures and pick the one that best performs on the *test set*. Estimates of performance of this

baseline are generous because we choose a combination that best performs on the *test set*. To be more conservative, we run the baseline with two sets of detectors (state-of-the-art models in [49] trained on our dataset, and state-of-the-art models in [48]) and pick the best one.

To evaluate our phrase detectors we test each of the visual phrase models and the corresponding baseline detector on a test set of approximately 200 images. Each test set has roughly 50 positive and 150 negative examples. The negative images are selected in a way that they do not contain any example of participating objects. For phrases that have only one participating object the baseline would be the corresponding models from [48].

Figure 3.4 depicts comparisons between the visual phrase detection results and the baseline. Note the significant improvements using visual phrase detectors trained on only 50 positive examples and default settings compared to heavily fine tuned object detectors [48] trained on thousands of examples. Further, the baseline is learned on the test set. Table 3.1 shows Average Precision (AP) for all of the visual phrase detectors compared with the results of the baseline detectors. In most cases our visual phrase detectors are outperforming the baseline detectors by significant margins despite the fact that the baseline is designed to perform best on the test set. There are visual phrases like "dog jumping" where neither the visual phrase detectors, nor the baseline detectors have promising results. These are hard objects and visual phrases with unmanageable variance in appearance. The results in Figure 3.4 and Table 3.1 support of the neglected fact that the appearance of the objects may change when they interact. Figure 3.4 and Table 3.1 show amazing gains when considering visual phrases.

3.4.2 Decoding

The role of the decoding algorithm is to encourage predictions that agree with other predictions and discouraging predictions that don't. For example, in Figure 3.2 the existence of a strong person riding horse detection has convinced our decoding algorithm to encourage a horse prediction and discourage two wrong person predictions. In our decoding method, visual phrases can improve

object detection and vice versa.

We compare our decoding algorithm with that of [43] on our phrase dataset. This is to evaluate our decoding method with other decoding methods not to evaluate the merits of phrasal recognition as all of our detectors, including visual phrase detectors, are provided as input to all decoding methods. We run all of the detectors for all of the phrases as well as the objects and construct the features as explained in section 3.3.1. We then use our decoding algorithm to learn a set of weights that rescore the confidences of the bounding boxes based on interactions. We use per class AP (classes include objects and visual phrases), overall AP and mean per image AP for comparisons. We also learn the model of [43] using the publicly available code on our dataset. We again rescore the confidences of the bounding boxes using the weights provided by this model and compute per class AP, overall AP and mean per image AP. All these three decoding procedures are learned on visual phrases as well as objects. Our decoding gets an overall AP of 0.319 and mean per class AP of 0.495 compared to the overall AP of 0.313 and mean per class AP of 0.493 for [43] and AP of 0.308 and mean per class AP of 0.491 for NMS using models in [49]. We believe that encoding the interactions in the representation makes the models more computationally manageable compared to encoding the interactions by pairwise terms in the model where inference is NP hard [43]. As a result, we don't need to use an approximation algorithm to sacrifice the accuracy or to find the exact minima and sacrifice the time; we run an *exact* inference algorithm which gives a better decoding performance.

3.4.3 Phrasal Recognition Helps Object Detection

We learn our decoding and the method of [43] using only the objects (not phrases) and compare it with the case when we consider both phrases and objects. Table 3.2 shows per class AP's for both our decoding and that of [43] with and without phrases. Significant gains in the performance of detectors when coupled with visual phrases establish the importance of visual phrases coupled with reliable decoding.

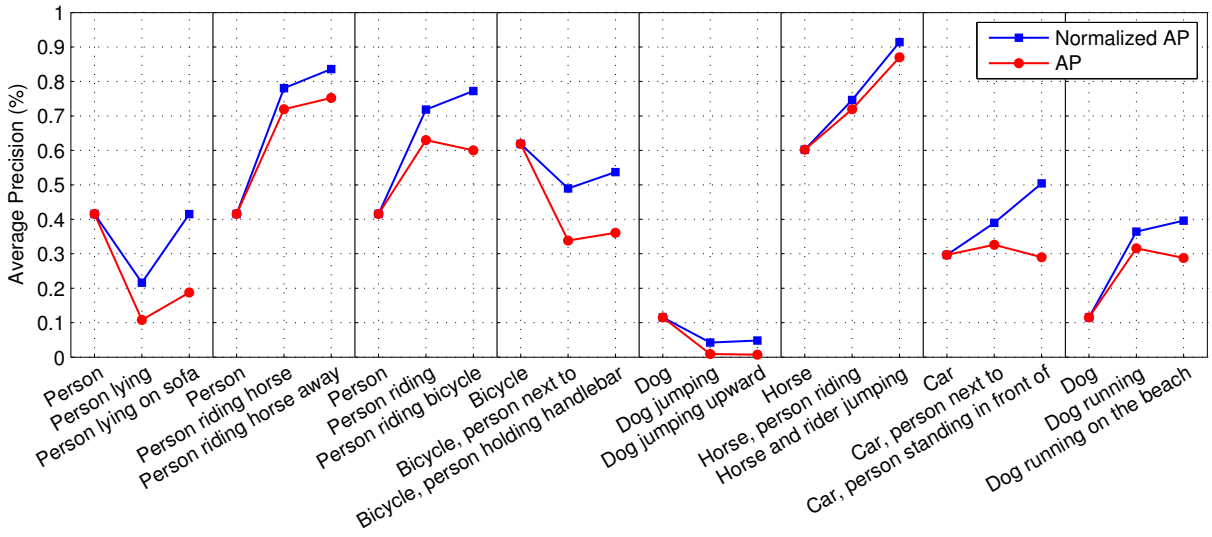


Figure 3.7: Phrasal recognition AP as the phrase becomes more specific. Since AP is significantly affected by the prior probability (chance) of the category in the test set, in addition to the regular AP we measure normalized AP from [71] as well. As the phrases become more specific the difference between the AP and the normalized AP becomes more significant suggesting visual phrases will suffer more from the prior probability as they become more specific. As a visual phrase becomes more specific sometimes the detection performance increases but sometimes it decreases. For example, "person riding horse" is easier to detect than "person", but "person lying on sofa" is harder. Note that this does not mean that "person lying on sofa" is a useless visual phrase, figure 3.4 shows a "person lying on sofa" detector significantly outperforms a general "person" detector when the goal is to detect "person lying on sofa".

Our decoding helps recognition of single objects using phrases. For example, in image "a" of Figure 3.6, a confident "person riding bicycle" detector helps to boost the bicycle detection and to suppress wrong person predictions. Object detections also help visual phrase recognition. For example, in image "b" of Figure 3.6, the highly confident sofa detector increases the confidence of the "dog lying on sofa" detections.

3.5 Analysis

Although phrasal recognition could significantly improve our ability to interpret images, not every construction of a visual phrase improves detection (Table 3.1).

Deng et. al. [67] study a number of challenges in image categorization given a large set of categories. They argue image categorization becomes harder a) given fine-grained categories, and further b) given similar categories. We have observed similar challenges in detecting visual phrases as well. In this section we investigate the challenges posed by a) fine-grained visual phrases and b) the similarity in visual phrases, and study a few success factors in constructing a visual phrase.

3.5.1 Challenges of Fine-Grained Visual Phrases

Fine-grained visual phrases naturally have fewer examples available than the generic objects. This could affect detection accuracy in two ways:

- The scarsity of positive examples in the *training set* could affect the detectors ability to learn a comprehensive model;
- The decreased prior probability of the category in the *test set* (or simply the *chance baseline*) directly affects the AP score.

There have been several works investigating the effects of scarcity in training examples on detection performance and suggesting various transfer-learning solutions. Zhu et. al. [68] show that the changes in the number of positive training examples could significantly affect the AP when dealing with few positive examples. Our experiments in figures 3.4 show improvement for the majority of visual phrases (given tens of positive examples). Lim et. al. [70] and Aytar et. al. [69] build detectors for scarse categories by borrowing examples from *similar* and *more common* categories. In a visual phrase setting this can be interpreted as borrowing examples from a more general visual phrase. This means if we design a too much specific visual phrase we would still need examples of a less specific visual phrase to successfully train the detector.

Hoiem et. al. [71] argue that Average Precision is sensitive to the prior probability of observing the category in the *test set*. Assume we compute the AP of a certain detector on two different

test sets: test set A with p positive examples and n negative examples, and test set B with the same p positive examples but $10n$ negative examples. Given a certain confidence threshold τ, the recall rate would be equal for both test sets (because they share the same set of positive examples), but the precision on B could be lower than on A, as B is likely to produce 10 times more false positives than A given the threshold τ (because it has 10 times more negative examples). Hoiem et. al. suggest *normalized Average Precision* is a more suitable measure for comparing detectors because it excludes the effects of the prior probabilitiy. Note that when comparing two different detectors for the same category on the same test set, we do not need to normalize the APs because the prior probabilities stay the same.

Figure 3.7 compares eight sets of increasingly specific visual phrases using both AP and normalized AP. The normalized AP's in this figure adjust the prior probability of each set to be equal to the prior probability of the left-most category in that set. As the phrases become more specific the difference between the AP and the normalized AP becomes more significant suggesting fine-tuned visual phrases will suffer from the prior probability. As a visual phrase becomes more specific sometimes the detection performance increases and sometimes it decreases. For example, "person riding horse" is easier to detect than "person", but "person lying on sofa" is harder. Note that this does not mean that "person lying on sofa" is a useless visual phrase, figure 3.4 shows a "person lying on sofa" detector significantly outperforms a general "person" detector when the goal is to detect "person lying on sofa". This suggests that constructing visual phrases could be an effective way to handle difficult subcategories.

3.5.2 Confusion with Similar Visual Phrases

Deng et. al. [67] have shown that discriminating between similar categories is harder than discriminating between uniformly sampled categories. Hoiem et. al. [71] show that confusion with similar categories has a devistating impact on the AP of current object detectors. Since different visual phrases are likely to share similar objects or actions, we study the performance effects of including

similar but *negative* examples (which we call *tough negative* examples) in both training and test sets.

We train two different detectors for each category using two different training sets (A and B) and test them on three different test sets (T_1, T_2 and T_3). Training sets A and B both contain 50 positive and 800 negative examples, however, B includes but A excludes tough negative examples. The three test sets each contain 50 positive and 100 negative examples; their difference is that T_3 includes tough negative examples, T_2 includes negative examples from other categories while T_1 excludes both. For each visual phrase we borrow tough negatives from the four most semantically similar but distinct visual phrases; for instance for the visual phrase "person next to horse" we borrow the members of "person next to car", "person next to bicycle", "person riding horse" and "person running".

Table 3.3 compares the APs for the training sets A and B and the three test sets T_1, T_2 and T_3. Figure 3.8 also compares the mean AP (over all categories) given all the combinations of the two training sets and the three test sets. This figure suggests that including tough negative examples in the training set does not actually improve the AP. However, it suggests that if our test set does not include any tough negative examples it would even be beneficial to exclude tough negative examples from the training set as well.

3.6 Related Work

Phrasal recognition is related to several different bodies of work and inspired by many important researches and have started new research directions. We study the prior work and the follow-up work related to phrasal recognition.

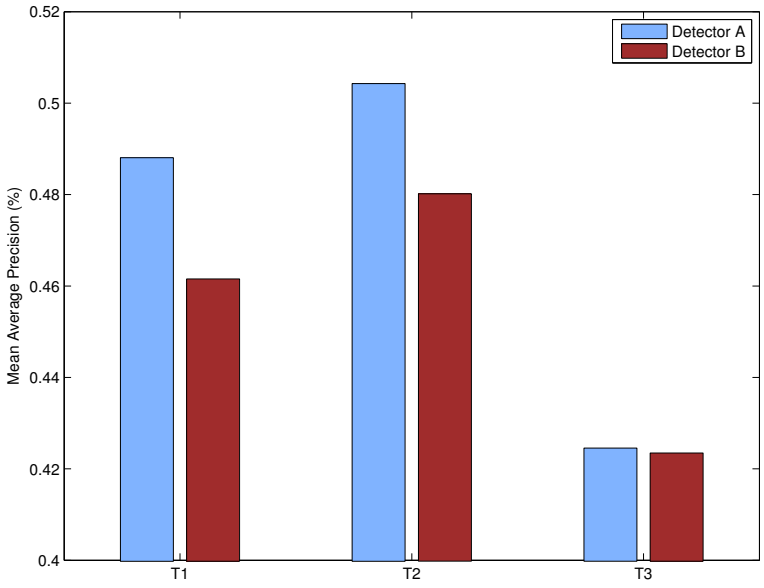


Figure 3.8: Comparison of the mean AP (over all categories) of the training sets A and B on the three test sets T_1, T_2 and T_3. According to the figure excluding tough negative examples from the training set would be helpful if the test set does not contain any tough negative examples. Furthermore including tough negative examples does not lead to any significant gain in handling tough negative examples in the test set.

3.6.1 Related Work

Object Recognition: Deformable templates [44, 45] and part based models [42, 50, 46] are of the most successful methods in object recognition. We use the standard version of the deformable part models (DPM) detectors in [49]. Our approach is independent of the choice of detectors; We use DPM because it produces state-of-the-art results.

Object Interactions: Prior to this work the changes in appearance of objects due to interactions with other objects were typically ignored in the methods that model interactions between objects. Gupta et. al. [12] model these interactions by modeling the prepositions and adjectives that relate nouns. Yao and Li [17] model the interactions between human pose and objects by coupling the human pose estimation and object recognition together. In [1] the interactions between objects is modeled implicitly in the context of predicting sentences for images. The most relevant to ours is the work by Desai, Ramanan, and Fowlkes [43]. They encode the interactions between objects by a set of relationships like "on the right of", "on the left of", "on the top of", etc. They

then learn a weight for the interactions of objects in each of these relationship bins and use them to re-weight the confidence of detectors. We differ from them as we consider the change in appearance of interacting objects. We show that neglecting the change in the appearance of interacting objects causes recognition issues, while modeling it significantly improves recognition results.

Scene Understanding has been one of the mainstream tasks in computer vision. One natural approach is to represent scenes as with global features that take into account general information about images [54, 52]. An alternative is to consider objects in the scene and discover clusters of correlated objects [53]. Objects in scenes are not independent and tend to cluster. We think these clusters might be formed at the phrase level as well. There is a neglected semantic gap between scenes and objects. We introduce visual phrases to cover this gap.

Machine Translation aims at automatic translation from one language to another one. Statistical translation methods are among successful approaches. In the common architecture of statistical translation models, there is a translation model, a language model, and a decoding algorithm. The decoding algorithm has to decide the final translation given the translation model, language model, and a query sentence. Word based translations are usually not desirable as there is no direct mapping between words across languages and syntactic differences are significant. However, phrasal translations, which are the inspirations of this work, are fashionable in machine translation because they allow multiple to multiple translations, use local context in translation, and allow translation of non-compositional phrases [51].

3.6.2 Subcategories

One way to cope with intra-class variations is to partition the data into smaller clusters, subcategories, where the intra-cluster variations are manageable. Different criteria have been explored to partition the data into subcategories. [49, 78] uses the aspect ratio of the labeled bounding boxes, [79] uses annotations of key points to find clusters of objects with very similar poses, [80, 81] use viewpoint information to obtain subcategories, [82, 83] use subordinate hierarchies and use

semantics to obtain subcategories, and [77] uses image similarities to obtain visually similar subcategories. Our work is related to subcategory recognition as our visual phrases are examples of subcategories. We differ from this body of work because we assume a non-exclusive model to deal with subcategories. A category is typically modelled as a mixture of subcategory models and the maximum scoring subcategory wins. This ignores the relationship between other subcategories. For example "person riding horse" should encourage "person" and "horse". Furthermore, in this mixture model horse doesn't have an entity, it is only used to estimate subcategory memberships. We believe some kind of intra-class variations can be modelled by the mixture models and some other kinds (visual phrases) should be encoded separately.

3.6.3 Single Image Activity Recognition

The problem of recognizing activity by just looking at a still image is studied in a number of works. This problem can be perceived as detecting a kind of visual phrase that correspond to a specific action of a person; For example "Person drinking from bottle" or "person jumping". Desai et. al. [20] makes use of pose-based object mixtures to detect sport activities, [84, 17] jointly reason about objects and their actions, [59] uses latent pose variables to encode activities, [61] study poselet activation patterns, and [60] uses latent SVM models with bag of words features and their combinations to recognize activities from a single images.

3.6.4 Visual Phrases

Visual phrases can be discovered by different criteria. Li et. al. use objects that tend to co-occur together to discover visual phrases [66]. They also show that visual phrases can be learned in a weakly supervised fashion, without bounding box annotation of objects [55]. Singh et. al. [72] extract discriminative patches that may correspond to either parts or visual phrases. Siyahjani et. al. [74] propose a context-aware sparse-coding scheme using an overcomplete dictionary

3.6.5 Interactions

Visual phrases encode complex interactions reliably. This suggests using them to model different kinds of interactions. Yang et. al. [56] model interacting people as visual phrases and show the advantages of their approach over modeling each person individually and reasoning about poses. Bilen et. al. [75] use visual phrases for image classification and show the windows introduced for the classification of class pairs improves their results. Pirsiavash et. al. [63] encode active objects in first-person videos as visual phrases. Desai et. al. [64] model the interactions as relational phraselets which are interesting combinations of visual phrases and poselets. [65] address the problem of image search and retrieval for joint-concepts.

3.6.6 Images and Sentences

Visual phrases can explain bigger chunks of an image with more information. This means that one can use visual phrases to generate sentences for images. [73] address the problem of sentence generation and assume a large chunk of the meaning of an image can be identified with a few descriptive phrases (keyphrases). They also propose a framework to generate sentences for images by extracting "keyphrases" from images. Feng et. al. [76] focuses on automatically generating captions for images given phrasal annotations.

3.7 Discussion

We introduce visual phrases and show the benefits of including them in the vocabulary of recognition. We show that visual phrases are much more reliable to detect and phrasal recognition helps object recognition. Our decoding algorithm offers improved recognition by learning the relations between visual phrases and objects. We perform detailed analysis of the effects of specificity in visual phrases, the reduction in visual complexity, and the separability of negative examples.

Our structural learning approach to decoding improves recognition of objects by learning the relations between objects and visual phrases. However, a simple SVM using our context-aware features provides slightly worse results suggesting that standard structural learning methods cannot leverage the relationships between visual phrases and objects. Future research involves devising better structure learning algorithms.

As the number of words increases, the number of possible visual phrases grows exponentially. This will eventually cause running time, training samples, and annotation issues. However, our experience of visual phrases mirrors the experience of machine translation community with linguistic phrases. The number of useful visual phrases (phrases) is significantly smaller than the number of all possible combinations of objects (words).

The dimensionality of our decoding features grows with the number of categories. However, there is no need to consider all of the categories when we model the interactions. For this reason, one might only consider a fixed number of related categories for each bounding box.

We speculate that the relations between attributes and objects, parts and objects, visual phrases and scenes, and objects and visual phrases mirror one another. Future work will investigate systems to decode complete sets of detections covering the semantic spectrum.

3.8 Follow-up Work

After the publication of visual phrases several recognition attempts were made that the focus of detection was shifted from detecting individual objects to visual phrase type patterns.

Singh et. al [72] studied discriminative patches as patches in the image that can classify a scene. These patches were detected in an unsupervised way. Li et. al [55] develop a method to identify groups of objects in an unsupervised way. They apply the new understanding to scene recognition. Desai et. al [43] identify interactions between objects in an unsupervised way. They call this Relational Phraselet and show that visual phrases can be accurately identified in an unsupervised

fashion.

Pirsiavash et. al [63] used visual phrases to identify the activities of people from first person cameras. Yao et. al [85] model learn to detect the interaction between humans and certain objects by modelling them as visual phrases. Choi et. al [86] develop 3D geometric phrases and use them to understand indoor scenes. They show that their model can effectively explain scene semantics, geometry and object groupings from a single image.

Fereshteh Sadeghi et. al [87] developed a visual knowledge extraction system that inputs a verb-based relation phrase between common nouns, and analyses the relation in a dataset of images. This system reasons about the spatial consistency of the relative configurations of the entities and the relation involved. Chen et. al [88] cluster images together with their textual description. This algorithm is able to automatically infer mappings between images and noun phrases. This mapping is not necessarily one-to-one. they can find multiple mappings for a given noun phrase.

Phrases (Trained with 50 positive images)	Phrase (AP)	Baseline (AP)	Gain (AP)
Person next to bicycle	0.466	0.252	**0.214**
Person lying on sofa	0.249	0.022	**0.227**
Horse and rider jumping	0.870	0.035	**0.835**
Person drinking from bottle	0.279	0.010	**0.269**
Person sitting on sofa	0.262	0.033	**0.229**
Person riding horse	0.787	0.262	**0.525**
Person riding bicycle	0.669	0.188	**0.481**
Person next to car	0.443	0.340	**0.103**
Dog lying on sofa	0.235	0.069	**0.166**
Bicycle next to car	0.448	0.461	**-0.013**
Dog Jumping	0.072	0.134	**-0.062**
Person sitting on chair	0.201	0.141	**0.060**
Person running	0.718	0.484	**0.234**
Person lying on beach	0.179	0.140	**0.039**
Person jumping	0.317	0.036	**0.281**
Person next to horse	0.351	0.287	**0.064**
Dog running	0.504	0.160	**0.344**

Table 3.1: AP scores for all of the visual phrases in our dataset. We compare our visual phrase detection results with a baseline detector that consists of the state of the art object detectors coupled with an operator that tries to best model the relationships between objects. This baseline is biased toward the best possible outcome on the test set. Please see section 3.4.1 for more details on the baseline. Note the significant gain (third column) in using visual phrases compared to an optimistic upper bound for detecting objects and modeling their relations. Some of the visual phrase detectors like "horse and rider jumping", "person riding horse", "person riding bicycle" show very significant gain. At the same time, some of the visual phrase detectors like "bicycle next to car" doesn't work as well. We demonstrate an opportunistic principle for selecting what detectors to use based on performance. See section 3.3.

	bicycle	bottle	car	chair	dog	horse	person	sofa
detectors of [48]	0.434	0.429	0.329	0.213	0.316	0.438	0.295	0.204
[43] without phrases	0.431	0.425	0.191	0.225	0.297	0.475	0.204	0.167
[43] with phrases	0.449	**0.435**	0.228	0.217	0.316	0.462	0.286	0.204
Our decoding -phrases	0.437	0.434	0.330	0.216	0.329	0.440	0.297	0.218
Our decoding +phrases	**0.457**	**0.435**	**0.344**	**0.227**	**0.335**	**0.485**	**0.302**	**0.260**

Table 3.2: Phrasal recognition helps object detection. This table compares the performance of our decoding with that of [43] with and without visual phrases using per class AP's. Adding visual phrases helps detection of objects. This table also shows that our decoding outperforms the state of the art object detectors of [48] and state of the art multiclass recognition method of [43].

Visual phrase	A on T_1	A on T_2	A on T_3	B on T_1	B on T_2	B on T_3
Bicycle next to car	0.5452	0.5562	0.4301	0.5311	0.5464	0.4313
Dog jumping	0.0796	0.1219	0.0862	0.1071	0.1374	0.0933
Dog lying on sofa	0.2500	0.2583	0.2528	0.1920	0.2098	0.1968
Dog running	0.5228	0.5215	0.4845	0.5263	0.5194	0.4791
Person next to bicycle	0.2784	0.3413	0.1999	0.3156	0.3642	0.3072
Person next to car	0.3627	0.3994	0.3205	0.3398	0.3590	0.3005
Person drinking from bottle	0.8834	0.8912	0.8813	0.8728	0.8706	0.8652
Person next to horse	0.1927	0.1954	0.1653	0.1584	0.1664	0.1547
Person jumping	0.3934	0.3621	0.3049	0.3510	0.3350	0.3180
Horse and rider jumping	0.7255	0.7345	0.5412	0.6238	0.6302	0.5284
Person lying in beach	0.5359	0.5738	0.4321	0.5239	0.5800	0.4851
Person lying on sofa	0.4965	0.5164	0.4289	0.4930	0.5217	0.4088
Person riding bicycle	0.7376	0.7405	0.6462	0.7088	0.7189	0.6740
Person riding horse	0.8150	0.8208	0.7780	0.8062	0.8209	0.7759
Person running	0.7511	0.7632	0.6346	0.6997	0.7063	0.5847
Person sitting on chair	0.3674	0.4106	0.3605	0.3094	0.3807	0.3476
Person sitting on sofa	0.3635	0.3691	0.2737	0.2906	0.2996	0.2517
Mean	0.4883	0.5045	0.4248	0.4617	0.4804	0.4237

Table 3.3: Average precision for various train/test configurations. The detectors of A are trained without any tough negative examples while the detectors of B are trained with tough negative examples. In all the three testing sets of T_1, T_2 and T_3 we use an identical set of 50 positive examples. T_1 contains 100 negative examples from the visual phrase dataset that contain none of the 20 standard PASCAL challenge objects. T_2 contains 100 negative examples from PASCAL VOC2008; the negative examples of T_2 contain no tough negative examples. T_3 contains 100 tough negative examples.

Chapter 4

Fast Object Detection using Vector Quantization

4.1 Introduction

A major burden in using object detectors in practice is speed. Except for certain objects including face, pedestrians and certain rigid objects, detectors do not currently run at video rate. We employ a series of techniques to detect several objects together at video rate. The architecture we present here can detect the 20 PASCAL VOC categories simultaneously at 30Hz.

We focus on speeding up DPM [49] because it is a mature and stable technology. While other detection methods are more accurate, the full potential of these technologies has not yet been explored, and they will not take their final form for some time. We believe that our speed-up techniques exploit fundamental properties of templates and will apply to deep leaning methods.

We build up our detector based on Deformable Parts Model [49] and compare to its latest implementation [108]. The latest implementation detects 20 PASCAL VOC object categories in about 13 seconds per image from the PASCAL dataset. Several techniques have been developed to speed up DPMs [89][98][4]. These techniques can speed up computation time to about 0.5 seconds (2Hz) with almost no loss of accuracy. Our techniques obtains a further speed-up to 30Hz with a minor loss of accuracy. Furthermore, our technique allow an explicit trade-off of accuracy for speed.

Furthermore, our technique can maintain a fixed frame rate at 30.0Hz that is essential in practical applications. Most speed-up techniques optimize average speed and cannot guarantee a fixed time per frame.

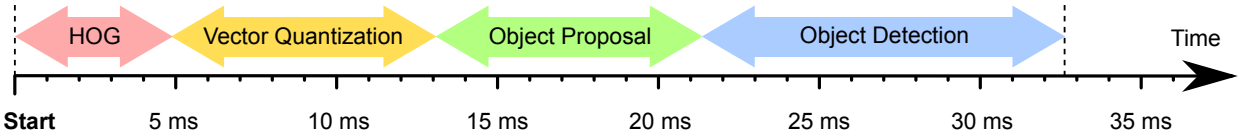


Figure 4.1: Our fast implementation of Deformable Parts Model can jointly detect 20 PASCAL categories at 30fps or faster. The pipeline consists of four steps that together run at video rate speed. To achieve this speed we used optimized techniques for each step. Optimizations for HOG feature computation are discussed in Section 4.4; Fast Vector Quantization is discussed in Section 4.5; The object proposal technique is discussed in Section 4.6; and object scoring is discussed in Section 4.7. For details about the exact computation time of our implementation please refer to Section 4.9. Allocation of time between the proposal and the detection phase can be balanced according to the processor architecture, dataset properties and application requirements; time allocation is discussed in Section 4.10.

Our technique consists of four major steps that are illustrated in Figure 4.1. Given a query image, we first extract a lightweight version of HOG features from the image (details in Section 4.4). We then vector quantize HOG features according to Section 4.5. We use a data-structure to obtain object proposal to identify the promising locations (Section 4.6). We finally score the proposals by evaluating the corresponding templates (Section 4.7).

We employ separate optimization techniques to speed up each of the four main stages. We implemented a highly optimized code to extract HOG features very quickly. Our implementation utilize various low-level optimizations including vector operations, multiple cores and CPU cache management. We also use a lightweight version of the HOG pyramid that further speeds up the process. After the HOG features are computed we use a hierarchical clustering process to vector quantize the HOG features (Section 4.5).

We use a data-structure to provide cheap object-dependent proposals in Section 4.6. Our proposal stage uses a hashing scheme that allows us to process only a small fraction of templates at each location. Our object scoring stage (Section 4.7) can also operate in a user-defined time-frame. It processes as many locations as it can within the specified time-frame.

Our implementation is fast and light-weight. The code is implemented in C++ but can be called from MATLAB. Our implementation is available for download at `vision.cs.illinois.`

edu/DPM30Hz. Our algorithm not only can process an image to detect 20 pascal categories simultaneously at 30fps, it can further trade off accuracy for time to achieve a detection rate of 100Hz.

4.1.1 Prior work

There is a rich literature of fast object detection built up on the original Deformable Parts Model [49] algorithm. Several successful speed-up techniques have been introduced in the last few years.

Cascades speed up evaluation by using rough tests to identify promising locations to further process using fine tests. One of the earliest successful examples was the face detection algorithm by Viola and Jones [92]. This algorithm runs crude tests for all locations and fine tests for the locations that have passed many previous tests. Felzenszwalb et al. [89] evaluate root models, and then evaluate the part scores only in promising locations. At each iteration their method evaluate the corresponding template only if the current score of the object is higher than a certain threshold. Sadeghi et al. [4] follow a similar approach but they use a fast vector quantization technique that is compatible with cascades to further boost the speed. Pedersoli et al. [99] estimate the score of a location using a lower resolution version of root templates and use higher resolution templates in high-scoring locations. Dollár et al. [97] enable neighbouring locations to communicate when a template is being evaluated. Cascade approach to object detection has been shown to be very successful for speed-ups.

Transform Methods evaluate templates at all locations simultaneously by exploiting properties of the Fast Fourier Transform. The advantage of these methods, pioneered by Dubout et al. [98] is that the computation is fast and exact at the same time. In comparison, most other techniques involve approximation. The disadvantage of this approach is that it is not *random-access*; a large chunk of the locations are processed in one pass making the algorithm incompatible with cascade techniques.

Hash Tables exploit locality sensitive hashing [102] to get a system that can detect many thousands of object categories in a matter of seconds [101]. This strategy appears effective and achieves a good speed-up with very large numbers of categories. Dean et al. [101] use a hash table at the core of their technique that allows them to spend computation for only the high-scoring locations. The advantage of this technique compared to cascades is that they don't require any computation for low chance locations whereas cascade algorithms examine every location at least once.

Vector Quantization is well-studied for data compression [105]. In the past few years several algorithms have used vector quantization to speed up computation. These techniques operate in situations where arithmetic accuracy is not crucial. Jégou et al. [104] successfully apply vector quantization to approximate nearest neighbour search. Kokkinos [100], Vedaldi et al. [103] and Sadeghi et al. [4] apply different variations of this approach to object detection and demonstrate significant speed-ups.

Hierarchical Classification techniques run multiple detectors in a tree structure with a depth of $\Theta(\log C)$ to be able to cover C categories. Nistér et al. [95] clusters categories using hierarchical k-means. Bengio et al. [111] use detectors that are suitable to discriminate between groups of categories. Both techniques are scalable in terms of C.

Object Proposals are used in object detection techniques that need to avoid a dense sliding window search. Some object proposal algorithms produce category-independent proposals (e.g. Endres et al. [93] and Cheng et al. [94]) while others [101] provide category-dependent proposals. The main source of speed-up in these techniques is that they significantly limit the number of locations to evaluate detectors. Category-dependent proposals are preferred in speed-up applications as they need to be evaluated by fewer detectors.

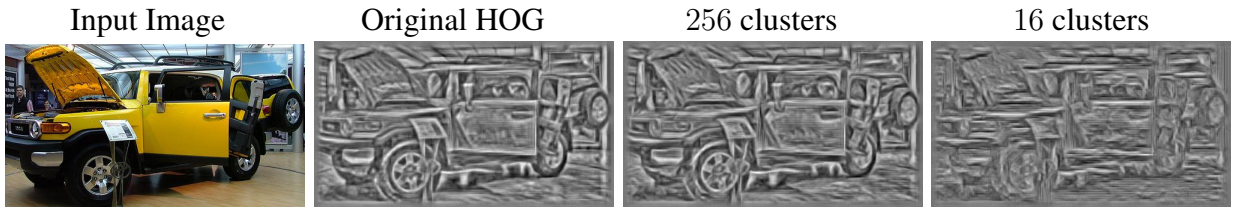


Figure 4.2: Visualization of Vector Quantized HOG features. (a) is the original image, (b) is the HOG visualization, (c) is the visualization of vector quantized HOG feature into $c = 256$ clusters, (d) is the visualization of vector quantized HOG feature into $c = 16$ clusters. HOG visualizations are produced using the inverse HOG algorithm from [**?**]. Vector quantized HOG features into $c = 256$ clusters can often preserve most of the visual information.

GPU Implementation can be used to speed up object detectors as well. Vanilla DPM [109] is a version of DPM that can harness the power of GPU to speed-up object detectors.

These techniques have improved the object detection speed so much that the feature computation stage has became a major bottleneck. Dollár et al. [90] present elegant techniques to speed-up features computation. We use a version of [90] to speed up our feature computation (Section 4.4).

4.2 Fast Approximate Scoring with Vector Quantization

The vast majority of modern detectors work as follows:

- In a preprocessing stage, an image pyramid and a set of underlying features for each layer of the pyramid are computed.
- For each sample point in each layer of the pyramid, a fixed size window of the image features spanning the sample point is extracted. A set of linear functions of each such window is computed. The linear functions are then assembled into a score for each category at that location.
- A post processing stage rejects scores that are (a) not local extrema and (b) under threshold.

Precisely how the score is computed from linear functions varies from detector to detector. For example, exemplar SVMs directly use the score; deformable part models summarize a score from several linear functions in nearby windows; and so on. The threshold for the post-processing stage is chosen using application loss criteria. Typically, detectors are evaluated by marking true windows in test data; establishing an overlap criterion to distinguish between false and true detects; plotting precision as a function of recall; and then computing the average precision (AP; the integral of this plot). A detector that gets a good AP does so by assigning high values of the score to windows that strongly overlap the right answer. Notice that what matters here is the ranking of windows, rather than the actual value of the score; some inaccuracy in score computation might not affect the AP.

In all cases, the underlying features are the HOG features, originally described by Dalal and Triggs [91]. HOG features for a window consist of a grid of cells, where each cell contains a d-dimensional vector (typically $d = 32$) that corresponds to a small region of the image (typically 8×8 pixels).

The linear function is usually thought of as an $m \times n$ table of vectors. Each entry of the table corresponds to a grid element, and contains a d dimensional vector $\mathbf{w}$. The score at location (x, y) is given by:

$$S(x, y) = \sum_{\Delta y=1}^{m} \sum_{\Delta x=1}^{n} \mathbf{w}(\Delta x, \Delta y) \cdot \mathbf{h}(x + \Delta x - 1, y + \Delta y - 1)$$

where $\mathbf{w}$ is a weight vector and $\mathbf{h}$ is the feature vector at a certain cell (both d-dimensional vectors). We wish to compute an approximation to this score where (a) the accuracy of the approximation is relatively easily manipulated, so we can trade-off speed and performance and (b) the approximation is extremely fast.

To do so, we quantize the feature vectors in each cell $\mathbf{h}(x, y)$ into c clusters using a basic k-means procedure and describe each quantized cell $q(x, y)$ using its cluster ID (which can range from 1 to c). Figure 4.2 visualizes original and our quantized HOG features. We pre-compute the partial dot product of each template cell $\mathbf{w}(\Delta x, \Delta y)$ with all $1 \leq i \leq c$ possible centroids and

store them in a lookup table $\mathbf{T}(\Delta x, \Delta y, i)$. We then approximate the dot product by looking up the table:

$$S'(x,y) = \sum_{\Delta y=1}^{m} \sum_{\Delta x=1}^{n} \mathbf{T}(\Delta x, \Delta y, q(x+\Delta x-1, y+\Delta y-1)).$$

This reduces the per template computation complexity of exhaustive search from $\Theta(mnd)$ to $\Theta(mn)$. In practice 32 multiplications and 32 additions are replaced by one lookup and one addition. This can potentially speed-up the process by a factor of 32. Table lookup is often slower than multiplication, therefore gaining the full speed-up requires certain implementation techniques that we will explain in the next section.

The cost of this approximation is that $S'(x,y) \neq S(x,y)$, and tight bounds on the difference are unavailable. However, as c gets large, we expect the approximation to improve. As figure 4.3 demonstrates, the approximation is good in practice, and improves quickly with larger c. A natural alternative, offered by Felzenszwalb et al. [89] is to use PCA to compress the cell vectors. This approximation should work well if high scoring vectors lie close to a low-dimensional affine space; the approximation can be improved by taking more principal components. However, the approximation will work poorly if the cell vectors have a "blobby" distribution, which appears to be the case here. Our experimental analysis shows vector quantization is generally more effective than principal component analysis for speeding-up dot product estimation. Figure 4.3 compares the time-accuracy trade-offs posed by both techniques.

It should be obvious that this VQ approximation technique is compatible with a cascade. As results below show, this approximate estimate of $S(x,y)$ is in practice extremely fast, particularly when implemented with a cascade. The value of c determines the trade-off between speed and accuracy. While the loss of accuracy is small, it can be mitigated. Most object detection algorithms evaluate for a small fraction of the scores that are higher than a certain threshold. Very low scores contribute little recall, and do not change AP significantly either (because the contribution to the integral is tiny). A further speed-accuracy tradeoff involves re-scoring the top scoring windows using the exact evaluation of $S(x,y)$. Our experimental results show that the described vector

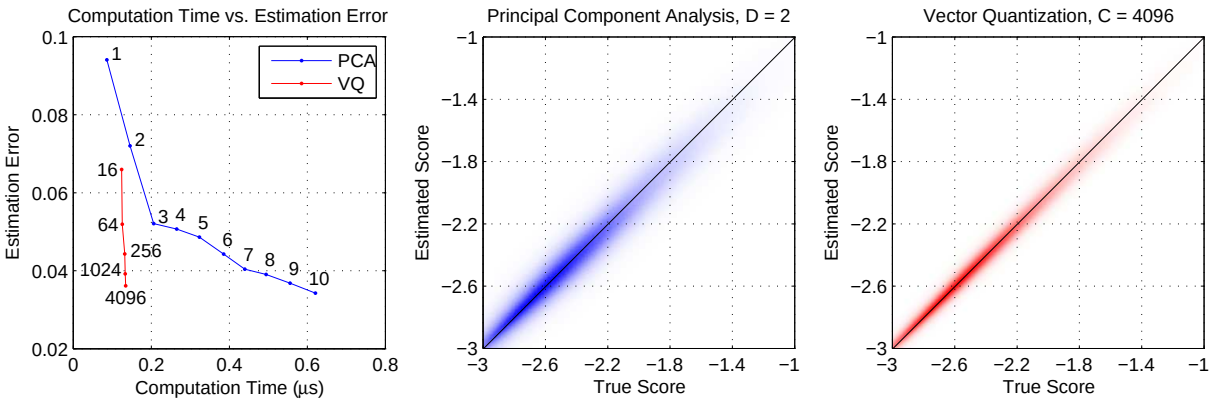


Figure 4.3: The left plot illustrates the trade-off between computation time and estimation error ($|S(x,y) - S'(x,y)|$) using two approaches: Principal Component Analysis and Vector Quantization. The time reported here is the average time required for estimating the score of a 12×12 template. The number of PCA dimensions and the number of clusters are indicated on the working points. The two scatter-plots illustrate template score estimations using 10^7 sample points. The working points $D = 2$ for PCA and $c = 4096$ for VQ are comparable in terms of running time.

quantized convolution coupled with a re-estimation step would significantly speed-up detection process without any loss of accuracy.

4.3 Fast Score Estimation Techniques

Implementing a vector quantization score estimation is straightforward, and is the primary source of our speedup. However, a straightforward implementation cannot leverage the full speed-up potential available with vector quantization. In this section we describe a few important techniques we used to obtain further speed.

Exploiting Cascades: It should be obvious that our VQ approximation technique is compatible with a cascade. We incorporated our vector quantization technique into the cascade detection algorithm of [89], resulting in a few folds speed-up with no loss of accuracy. The cascade algorithm estimates the root score and the part scores iteratively (based on a pre-trained order). At each iteration it prunes out the locations lower than a certain score threshold. This process is done in two passes; the first pass uses a fast score estimation technique while the second pass uses the

original template evaluation. Felzenswalb et. al. [89] use PCA for the fast approximation stage. We instead use vector quantization to estimate the scores. In the case of deformable parts model this procedure limits the process for both convolution and distance transform together. Furthermore, we use more aggressive pruning thresholds because our estimation is more accurate.

Fast deformation estimates: To find the best deformation for a part template, Felzenswalb et. al. [89] perform an exhaustive search over a 9×9 grid of locations and find the deformation $(\Delta x, \Delta y)$ that maximises

$$\max_{\Delta x, \Delta y} S(\Delta x, \Delta y) = S_{app}(\Delta x, \Delta y) + S_{def}(\Delta x, \Delta y) \qquad -4 \leq \Delta x, \Delta y \leq 4$$

where S_{app} is the appearance score and S_{def} is the deformation score. We observed that since S_{def} is convex and significantly influences the score, searching for a local minima would be a reasonable approximation. In a hill-climbing process we start from $S(0, 0)$ and iteratively move to any neighbouring location that has the highest score among all neighbours. We stop when $S(\Delta x, \Delta y)$ is larger than all its 8 neighbouring cells (Figure 4.4). This process considerably limits the number of locations to be processed and further speeds up the process without any loss in accuracy.

Packed Lookup Tables: Depending on the detailed structure of memory, a table lookup instruction could be a couple of folds slower than a multiplication instruction. When there are multiple templates to be evaluated at a certain location we pack their corresponding lookup tables and read them all in one memory access, thereby reducing the number of individual memory references. This allow using SIMD instructions to run multiple additions in one CPU instruction cycle.

Padding Templates: Packing lookup tables appears unhelpful when there is only one template to be evaluated. However, we can obtain multiple templates in this case by zero-padding the original template (to represent various translates of that template; Figure 4.4). This allows packing

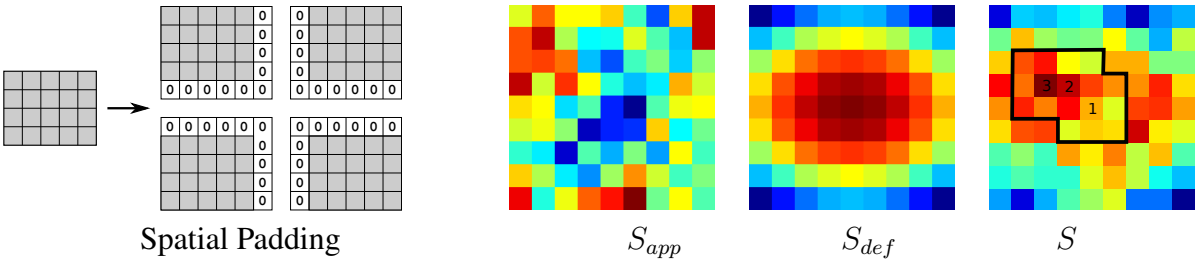


Figure 4.4: Left: A single template can be padded spatially to generate multiple larger templates. We pack the spatially padded templates to evaluate several locations in one pass. Right: visualization of S_{app}, S_{def} and S. to estimate the maximum score we start from center and move to the highest scoring neighbour until we reach a local maximum. In this example, we take three iterations to reach global maximum. In this example we compute the template on 17 locations (right image).

the lookup tables to obtain the score of multiple locations in one pass.

Sparse lookup tables: Depending on the design of features and the clustering approach lookup tables can be sparse in some applications. Packing p dense lookup tables would require a dense $c \times p$ table. However, if the lookup tables are sparse each row of the table could be stored in a sparse data structure. Thus, when indexing the table with a certain index, we just need to update the scores of a small fraction of templates. This would both limit the memory complexity and the time complexity for evaluating the templates.

Fixed point arithmetic: The most popular data type for linear classification systems is 32-bit single precision floating point. In this architecture 24 bits are specified for mantissa and sign. Since the template evaluation process in this work does not involve multiplication, the power datum would stay in about the same range so one could keep the data in fixed-point format as it requires simpler addition arithmetic. Our experiments have shown that using 16-bit fixed point precision speeds up evaluation without sacrificing the accuracy.

4.4 Pyramid of Features vs. Pyramid of Templates

Conventional object detectors operate at various scales to be able to detect objects with variable sizes. Template based object detectors extract local features at various scales (e.g. HOG pyra-

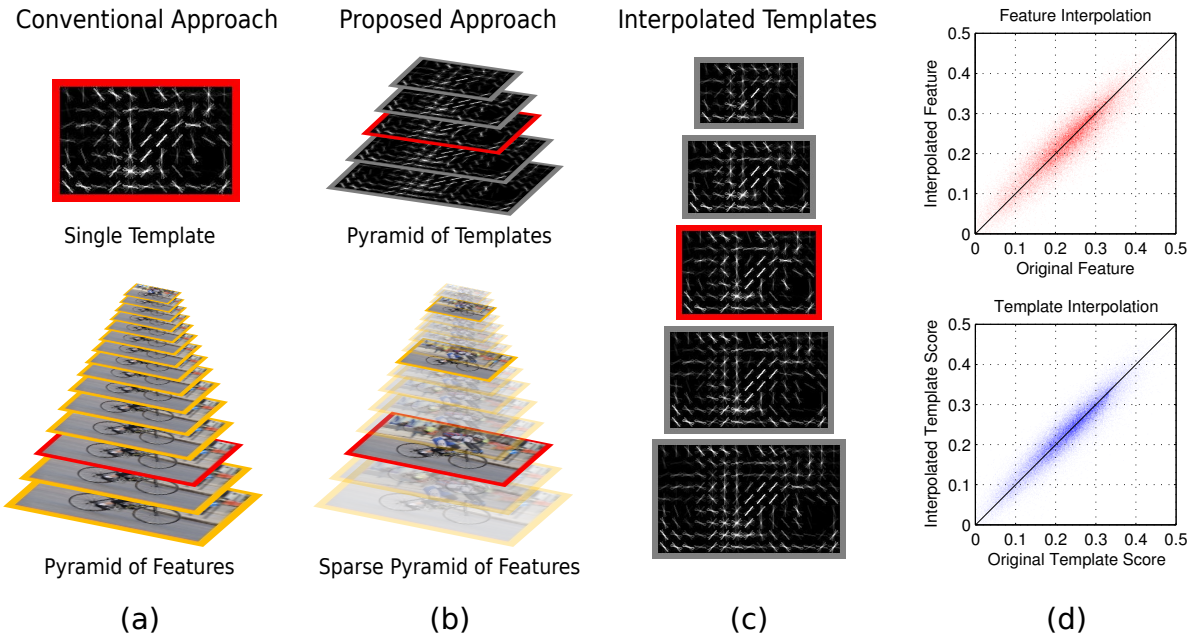


Figure 4.5: **a**: Conventional object detectors run templates on a pyramid of features to capture a range of scales (ten scales per octave is typical). Dollár et al. [90] compute two scales per octave then interpolate the rest of the scales to considerably speed up feature computation at the cost of about 2% loss of average precision. **b**: Instead of interpolating features we interpolate templates. We show that interpolating templates is faster and leads to further speed-up techniques. **c**: We generate new templates by interpolating templates to different scales. **d**: This process introduces some error. The two scatter plots illustrate original template score versus the score produced by interpolated features/templates. **d**: **Top**: Features are interpolated according to [90]. **Bottom**: Templates are interpolated instead of features. Although interpolating templates is faster than interpolating feature pyramids, the errors are in the same range.

mid [91] and histogram of sparse codes [106]) and evaluate a given template at all scales. In practice ten scales per octave is typical.

Feature computation is a major bottleneck for pedestrian detection. Dollár et al. [90] present an elegant technique to process features for certain key scales (one or two per octave) and interpolate for the rest of scales. Their experiments with pedestrian detection show that this leads to a significant speed-up for feature computation. Benenson et al. [112] interpolate templates for integral channel features. Our approach is similar to Dollár et al. [90]; however, instead of rescaling features we rescale templates (Figure 4.5). We rescale each template to several scales in order

to make a *Pyramid of Templates*. Our experiments show that this works as accurate as rescaling features while being faster in practice. The pyramid of templates has two major advantages over the regular pyramid of features:

1. Because HOG templates are several times smaller in size than HOG features (2K cells per object category vs. 100K cells per image) processing and storing HOG features takes much more time than templates. Furthermore, categories are often fixed for several images while every new image comes with a new feature. As a result, reducing the number of feature levels per octave directly limits the space required to store features. In our experiments HOG features are compressed from 8MB to 1.6MB per image. The benefits include more efficient caching and more efficient mobile application.

2. Several speed-up techniques are based on having a large number of templates (e.g. Pirsiavash et al. [96] and Dean et al. [101]). The computational complexity of [101] is claimed to be independent of C the number of templates. Their computational complexity depends only on L the number of locations to evaluate templates (whereas most algorithms have at least a $\Theta(CL)$ term in their complexity). Our technique uses $5C$ templates and $\frac{1}{5}L$ locations; therefore, it can directly benefit from speed-up techniques presented by Pirsiavash et al. [96] and Dean et al. [101].

A few technical issues arise when resizing templates for object detection. All part templates need to be resized as well as deformation costs and part locations. The interpolation method can affect the quality of the new template. In order to resize a HOG template we interpolate every layer separately. We compared bilinear and bicubic interpolations and bicubic interpolation appears to be the best. The interpolated weights are adjusted by a factor to maintain the mean and the standard deviation of the scores. Our experiments show that this optimization leads to an mAP loss of about 0.02, compatible to that of [90].

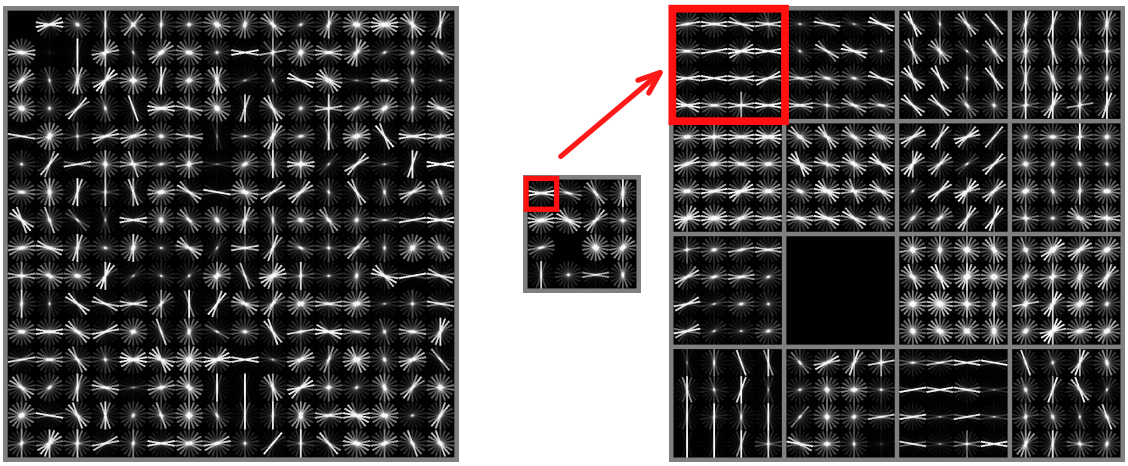


Figure 4.6: **a**: The method proposed by Sadeghi and Forsyth [4] quantizes each cell into one of 256 pre-defined clusters. Nearest neighbour search is a significant bottleneck in their technique. We use hierarchical clustering instead of flat clustering. **b**: each cell is first quantized into one of the 16 clusters. **c**: Depending on the first level, the cell is clustered into one of 16 clusters in the respective group in c. Note that hierarchical clustering reduces the number of comparisons from 256 per cell to two stages of 16 comparisons per cell.

4.5 Hierarchical Vector Quantization

Several optimization techniques have been employed to speed up Deformable Parts Model object detectors. The fastest was proposed by Sadeghi and Forsyth [4]. This is nearly two orders of magnitude faster than the original implementation of [108]. The key to their success is a vector quantization technique that decreases the computation demand by a large factor. They vector quantize HOG features and compute template scores by indexing certain look-up tables and adding their scores.

We use vector quantization for the same purpose but with a slightly different approach. The main computation bottleneck in [4] is vector quantization. They need 70ms per image to quantize HOG features for one image. The high computational demand is due to the fact that each HOG cell needs to be compared against every one of 256 cluster centers. (Figure 4.6, a). We use a hierarchical clustering technique to speed up this process. We first cluster each cell into 16

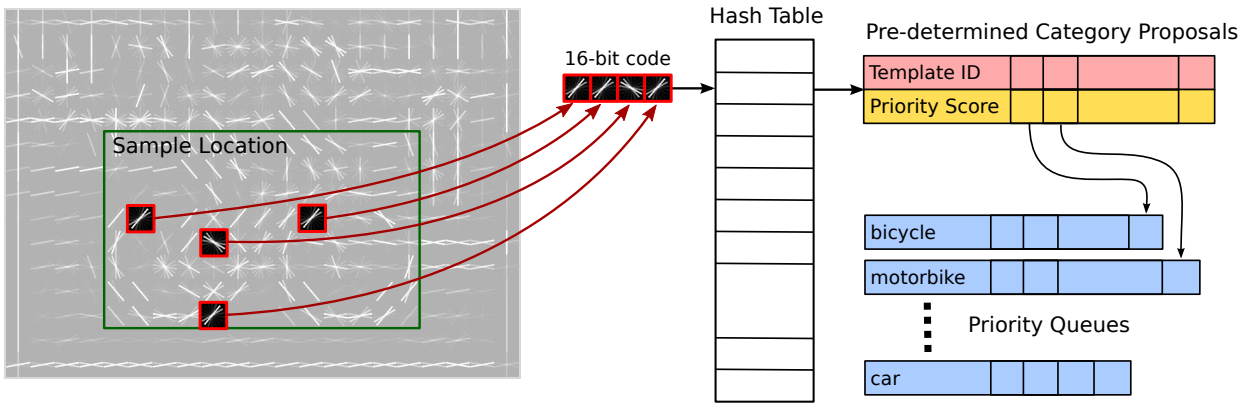


Figure 4.7: Our proposal generation data-structure. We use a few look-up tables that are filled with pre-determined proposals. For each location we make a hash code by observing four pre-specified cells. We index the code into a hash table and obtain a list of pre-determined category proposals for each location. We store the proposals in category specific priority lists and later use them to evaluate the score of each location for each proposed category.

clusters (Figure 4.6, b). Then according to the nearest cluster in the first step we compare against 16 other clusters to find the nearest cluster (Figure 4.6, c). We pre-compute clusters using k-means algorithm.

Our experiments show that the proposed hierarchical clustering technique leads to a negligible loss of 0.001 in mAP. In contrast, the speed-up gain is about 8-fold.

4.6 Object Proposal using Hash Table

We cannot evaluate all templates at all locations fast enough. Instead, we use a hashing technique to identify promising locations and insert them into priority queues (Figure 4.7). Proposals will then be processed in the object scoring stage (Section 4.7). This architecture means that both the proposal process and the template evaluation process can be terminated at any time allowing our method to operate at fixed frame rate and trade-off accuracy for speed.

The cascade framework applied to Deformable Parts Model first evaluates a rough version of

a given root template and then evaluates the corresponding part scores and finally re-estimate the scores by using fine templates (e.g. Felzenszwalb et al. [89] and Sadeghi et al. [4]). Although cascade methods prune the majority of locations, they need to at least evaluate all root templates at all locations (they can prune part templates but not root templates). To process the 20 PASCAL categories this step takes about 400ms in [89] and about 90ms in [4]. The two techniques are both too slow for video rate speed.

Several algorithms are introduced to generate object proposals for object detection (e.g. Endres et al. [93] and Cheng et al. [94]). We use a proposal generation data-structure to limit template evaluation to a sparse set of proposals rather than dense sliding window search. Our data-structure uses a hash table similar to Dean et al. [101]. Our hash table is distinguished from [101] in three aspects:

1. Instead of Winner Takes All (WTA) hash we use a hashing scheme compatible to our hierarchical vector quantization (Figure 4.7).

2. The data-structure used by Dean et al. [101] proposes template ID's without proposed scores. Our data-structure provides a *priority score* with each template ID. The priorities help us later choose which templates to process further.

3. The data-structure used by Dean et al. [101] uses hundreds or thousands of hash tables. We instead use 10 hash tables. The fact that we need fewer hash tables is partly due to our pre-stored priority scores.

4.6.1 Hash Codes

In order to generate proposals, we process all locations in an image (sliding window search) using our data-structure. Since different templates are different in size, we refer to each location using its top left co-ordinate (Figure 4.7). At each location we extract proposals using 10 separate hash tables.

Each hash table is indexed with a distinct 16-bit code. The 16-bit code is generated by observing four quantized cells and concatenating their corresponding quantization ID. We use a dictionary of 16 words (4-bits) for each cell that is equivalent to the first level of hierarchical quantization discussed in Section 4.5 (Figure 4.6). Reference cells are randomly determined for each hash table while initializing the data-structure.

Each cell of the hash tables is linked to a list of proposed templates and their corresponding priorities. A template can be determined by its category, root index and scale. For the PASCAL dataset and Deformable Parts Model version 5, there are 20 categories, 6 root templates per category and 5 scales (Figure 4.5): a total of 1200 root templates. We store 20 templates for each cell of the hash table. However, most of the proposals are not used in most cases.

4.6.2 Priority Lists

For each root template we store a separate priority list (Figure 4.7). Each list stores several proposal locations with their corresponding priority scores. The priority scores are used to determine the priorities between locations given a root template. Each root template has a limited budget of locations to examine that are chosen according to priority scores.

We use a simple array to store each priority list. After the lists are populated with proposals we keep a number of proposals that are expected to complete within the specified time-frame. We then evaluate root templates on remaining locations and update their priority score with the actual responses from the root templates.

The reason we store a separate priority list per template – as opposed to one joint priority list – is that the scores of different root templates are not directly comparable. Also the user may need to specify more process time on a certain template depending on the application. In our experiments we process equal number of locations per root template. Process allocation could be adjusted depending on the architecture of processor and the application. We discuss time allocation in more

detail in Section 4.10.

4.6.3 Hash Table Initialization

We use 10 parallel hash tables each indexed with a separate 16-bit code. A hash code is generated by concatenating the quantization ID's of four cells. We randomly choose the cells in a 12×12 window and build the look up tables accordingly. For each possible hash code we compute a rough approximation score using the look-up tables used by Sadeghi and Forsyth [4]. We choose the 20 top categories according to the approximation scores. We perform a score adjustment process to make sure all templates are equally likely to be proposed.

We process the 10 hash tables in a sequence to balance proposals among all locations in case of early termination. If not enough time is available to go through all hash tables, the tables used will cover image locations fairly.

4.7 Object scoring

Our proposal generation process provides a separate priority queue for each template. Because the number of proposals is often more than what we afford to process, many proposals cannot be evaluated.

We use a version of Round-Robin algorithm to process priority queues corresponding to different templates. We process one location from each queue in a circular order, handling all queues with equal priority. As soon as one proposal from one queue is done we process an example from the next queue. We continue this process until time is out. Our algorithm is parallelized with OpenMP to harness the power of all processor cores. Each thread in our process is responsible for an equal number of templates. All threads stop when time is up and return their detections. The time required for Non-max suppression (NMS) is also negligible.

We follow the technique presented by Felzenszwalb et al. [89] to process each location. Given a proposal, we first approximate the score of the root template using FTVQ [4]. We then add the approximated score of the first part together with its deformation cost. We continue adding the score of all other parts in a sequence. After we evaluate a part score we may stop and reject the proposal according to a pre-trained threshold.

After computing the approximated scores using FTVQ, we replace the approximated score of the root template and the part templates with their exact score in the same order. Again we may stop the process in each step according to a threshold. If the proposal is able to pass all steps we may report it according to NMS results.

Felzenszwalb et al. [49] and Sadeghi et al. [4] cache part template scores in their implementation to avoid re-computation. Because we operate in a sparse set of locations, the chances that a part template score is re-used at a certain location is small. Therefore we don't cache any scores. We observed that not caching scores could improve speed in our implementation as we need to allocate lower memory so we can utilize hardware cache more effectively.

4.8 Computation Cost Model

In order to assess detection speed we need to understand the underlying computation cost. The current literature is confusing because there is no established speed evaluation measure. Dean et al. [101] report a running time for all 20 Pascal VOC categories at the same time including all the preprocessing. Dubout et al. [98] only report convolution time and distance transform time. Felzenszwalb et al. [89] compare single-core running time while others report multi-core running times.

Computation costs break into two major terms: per image terms, where the cost scales with the number of images and per (image$\times$category) terms, where the cost scales with the number of

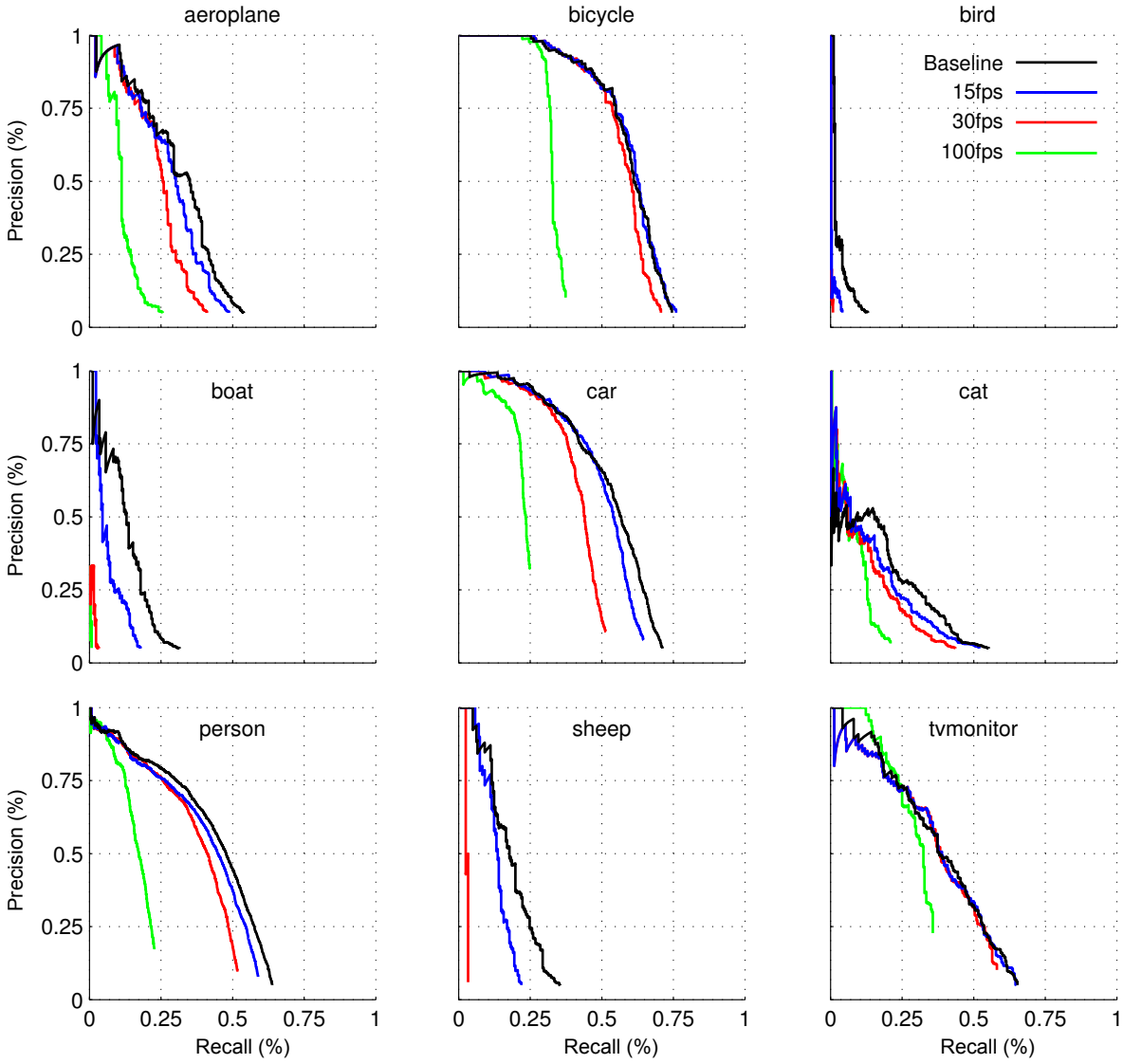


Figure 4.8: Precision-Recall curves for 9 objects in PASCAL dataset comparing to the baseline. The black curve (above) corresponds to the accuracy of deformable parts model at regular speed (Table 4.1). In the blue curve all 20 PASCAL categories are detected at once in a time frame of 67ms (15fps). In the red curve all 20 PASCAL categories are detected at once in a time frame of 33ms (30fps). In the green curve all 20 PASCAL categories are detected at once in a time frame of 10ms (100fps). For all precision recall curves a threshold is chosen so each PR curve would cover precision > 0.05. In practical applications often one working point is chosen in the high precision area. Note that the gap between the curves in the high precision are tiny within the red, the blue and the black curves. This means in applications where a high precision working point is set, the loss is less noticeable. Note that the green curve fails to produce any detections for bird, boat and sheep categories. More information about APs can be found in Table 4.1.

categories as well as the number of images. The total time taken is the sum of four costs:

- **Computing HOG features** is a mandatory, per image step, shared by all HOG-based detection algorithms.
- **per image preprocessing** is any process on image data-structure except HOG feature extraction. Examples include applying an FFT, or vector quantizing the HOG features.
- **per category preprocessing** establishes the required detector data-structure. This is not usually a significant bottle-neck as there are often more images than categories.
- **per (image×category) processes** include convolution, distance transform and any post-process that depends both on the image and the category.

Table 4.2 compares the performance of our approach with four major state-of-the-art algorithms. Template-based object detection is embarrassingly parallel by nature. The algorithms described are evaluated on various scales of the image with various root templates. We compared algorithms based on parallel implementation. Reference codes published by the authors (except [89]) were all implemented to use multiple cores. We parallelized [89] and the HOG feature extraction function for fair comparison. We evaluate all running times on a XEON E5-1650 Processor (6 Cores, 12MB Cache, 3.20 GHz).

4.9 Experimental Results

To evaluate our algorithm we compare it to a set of algorithms that are all based on Deformable Parts Model [49]. We evaluate our algorithm with three frequency settings: 15fps, 30fps and 100fps. We compare the techniques on PASCAL VOC 2007 that is established as a standard baseline.

We evaluated our algorithm by looking at the detection time and average precision (AP) score with respect to our baseline. We use DPM V5 [108] as our Average Precision baseline that is the

most recent and most accurate implementation of DPM. To evaluate the time we compare to [4] that runs nearly two orders of magnitude faster than [108] and is the fastest algorithm before this publication. Our algorithm run on a system with an Intel Xeon E5-1650 processor and 32GB of RAM. Both our proposed algorithm and the baseline utilize all the 6 cores of the CPU at full load.

Our algorithm runs legacy models from DPM V5 [108] that are trained to have 6 root templates per category and 8 parts per root template. Our algorithm doesn't need to train a new model, we build up our model by processing the pre-trained detectors of DPM V5 [108].

We use a separate optimization techniques to speed up each of the stages of [4]. We implemented a highly optimized function to extract HOG features very quickly. Our implementation uses AVX vector operations and multiple cores. It is also optimized to utilizes CPU cache carefully. We also limit the number of layers to extract HOG features by a factor of 5. These optimizations together speed up HOG feature computation from 40ms in [4] to an average of 4ms.

We use a hierarchical clustering process to vector quantize HOG features (Section 4.5). Our hierarchical algorithm examines two sets of 16 clusters per HOG cell that is 8 times lower than that of [4] which examines 256 clusters in one layer. Since we process five times fewer feature layers (as mentioned in Section4.4), the average vector quantization load is further reduced. The total time required for our vector quantization technique is down from 70ms in [4] to about 5ms on average.

Our object proposal stage (Section 4.6) allows us to process only a small fraction of templates at each location. It can terminate early to acomodate time for other stages. Our object proposal process will terminate in 7ms if it is not terminated early. Our object scoring stage (Section 4.7) can also operate in a specified time-frame. On average it takes about 1.2μs to process one location for one category (including root and part scores). This algorithm processes as many locations as it affords in the specified time-frame. In the fastest case it can run at 100Hz. In this speed our algorithm affords to process only one location per root template (For PASCAL 2007 we have 1200

root templates that is 20 categories $\times 6$ components $\times 5$ scales).

Our implementation is very fast and light-weight. The code is implemented in C++ but can be called from MATLAB. Our algorithm can process an image to detect 20 pascal categories simultaneously at 30fps or faster. It can further trade off accuracy for time; it can run at 100Hz while detecting 20 PASCAL categories in a time frame of 10ms. It also requires less than 10MB of memory at its peak demand to process an image for 20 categories (PASCAL Images are mostly 350×500 pixels large). This is three orders of magnitude faster than the original DPM V5 [108] implementation that itself is highly optimized.

Table 4.1 compares our algorithm with two established baselines. Our algorithm achieves 30Hz with an mAP of 0.26. At 15Hz, its mAP is 0.30; and at 100 Hz, its mAP is 0.16. Frequency is computed as $\frac{1}{t}$ where t is the time to detect all the 20 PASCAL VOC categories together in one image. This time includes features computation time but excludes the time to load image. We exclude the time to load the image because the time highly depends on the media.

Precision-Recall curves for our experiments are illustrated in Figure 4.8. Note that the gap between the curves in the high precision area in tiny between the red, the blue and the black curves. This is very important in practical applications as they often consider false positives costly and work in high precision regimes.

Table 4.3 compares our algorithm to several variations of DPM in terms of speed and accuracy. We report running time to detect all 20 PASCAL categories from raw image. We also compare our mean Average Precision to other techniques. In this table we compared to only algorithms that run on CPU. The fastest algorithm on GPU is Vanilla DPM [109] that runs at about 1Hz to detect the 20 PASCAL categories in a 640×480 image. It cannot sacrifice accuracy for speed.

Our algorithm can trade off accuracy for speed. Figure 4.9 illustrates the trade-off for both detecting all objects jointly and also detecting only a single object. This figure shows some detectors fail at 100fps while some others remain robust.

4.9.1 Exemplar Detectors

Exemplar SVMs are important benchmarks as they deal with a large set of independent templates that must be evaluated throughout the images. We first estimate template scores using our vector quantization based library. For the convolution we get roughly 25 fold speedup comparing to the baseline implementation. Both our library and the baseline convolution make use of SIMD operations and multi-threading. We re-estimate the score of the top 1% of locations for each category and we are virtually able to reproduce the original average precisions (Table 4.4). Including MATLAB implementation overhead, our version of exemplar SVM is roughly 8-fold faster than the baseline without any loss in accuracy.

4.10 Discussion

We believe that there are further improvements available. We expect that speed could be improved by exploring our hashing process to: (a) interleave image loading and feature computation; and (b) avoid feature computation at some image blocks. We expect accuracy could be improved by careful tuning of time allocation (a) between proposal and detection process and (b) between templates.

The trade-offs in Figure 4.9 shows some detectors fail at 100fps while some others remain robust. This suggests the optimal time allocation is not to allocate equal time to each category; some categories need more time while some categories need less.

The optimal time allocation depends on several factors including: processor architecture, the global time limit, the demand by each category and the application defined priorities for detecting different categories. Feature extraction and quantization require a fixed processing budget. Our design allows the rest of the budget to be divided between proposal generation and object scoring. The optimal partition depends on the application.

Our experiments show objects that are harder to detect suffer more with a limited budget (see Figure 4.8, boat, bird) whereas categories with higher AP remain more robust. Furthermore, Certain objects are more likely to appear in groups (e.g. sheep, person) so they are more sensitive to limiting the number of locations to process. The study of optimal process allocation in different situations requires an extensive study that doesn't fit into the context of this work.

Our trade-off allows for any speed improvement technique to directly result in accuracy improvement. The choice of working point in speed-accuracy trade-off allows for further data such as video or depth to be used for speed or accuracy.

Method	Ours	Ours	Ours	FTVQ [4]	DPM V5 [108]
Frequency	**100Hz**	**30Hz**	**15Hz**	**2Hz**	**0.07Hz**
aeroplane	0.1630	0.2695	0.3029	0.3320	0.3318
bicycle	0.3563	0.5735	0.5946	0.5933	0.5878
bird	0.0021	0.0909	0.0909	0.1027	0.1019
boat	0.0303	0.0303	0.1141	0.1568	0.1801
bottle	0.0909	0.1938	0.2425	0.2664	0.2535
bus	0.2989	0.4130	0.4720	0.5129	0.5056
car	0.2505	0.4240	0.4996	0.5373	0.5271
cat	0.1368	0.1725	0.1931	0.2251	0.1904
chair	0.0909	0.0909	0.1053	0.2010	0.2046
cow	0.0909	0.1062	0.1994	0.2432	0.2444
diningtable	0.1743	0.2500	0.2510	0.2685	0.2750
dog	0.0507	0.1159	0.1159	0.1260	0.1238
horse	0.2724	0.4735	0.5539	0.5651	0.5709
motorbike	0.2019	0.3850	0.4399	0.4849	0.4838
person	0.1962	0.3736	0.3971	0.4322	0.4327
pottedplant	0.0909	0.1179	0.1129	0.1345	0.1366
sheep	0.0000	0.0909	0.1702	0.2085	0.2154
sofa	0.1208	0.2860	0.3497	0.3568	0.3633
train	0.2801	0.3962	0.4198	0.4520	0.4651
tvmonitor	0.3075	0.3703	0.3840	0.4216	0.3943
mean AP	0.1603	0.2612	0.3004	0.3310	0.3294

Table 4.1: Comparison of different frame rates of our method with two major implementations of Deformable Parts Model: Fast Template evaluation using Vector Quantization (FTVQ) [4] and Deformable Parts Model (DPM) Version 5 [108]. We report per category AP that is computed as the average of precisions at 11 recall rates. Frequency is computed as $\frac{1}{t}$ where t is the time to detect all the 20 PASCAL VOC categories in one image. This time includes features computation time but excludes the time to load the image. We compare the algorithms on PASCAL VOC 2007 challenge that is a standard for benchmarking detection performance. Precision-Recall curves are illustrated in Figure 4.8.

	HOG features	per image	per (image×category)	per category
Original DPM [49]	40ms	0ms	665ms	0ms
DPM Cascade [89]	40ms	6ms	84ms	3ms
FFLD [98]	40ms	7ms	91ms	43ms
Our+rescoring	40ms	76ms	21ms	6ms
Our-rescoring	40ms	76ms	9ms	6ms

Table 4.2: Average running time of the state-of-the-art detection algorithms on Pascal VOC 2007 dataset.

Method	mAP	time
HSC [106]	0.343	180s
WTA [101]	0.240	26s
DPM V5 [108]	0.330	13.3s
DPM V4 [107]	0.301	13.2s
DPM V3 [49]	0.268	11.6s
Vedaldi et al. [103]	0.277	7s

Method	mAP	time
FFLD [98]	0.323	1.8s
DPM Cascade [89]	0.331	1.7s
FTVQ [4]	0.331	0.53s
Ours at 15Hz	0.300	0.07s
Ours at 30Hz	0.261	0.03s
Ours at 100Hz	0.160	0.01s

Table 4.3: Comparison of various versions of DPM [49]. The reported time here is the time to complete the detection of 20 categories starting from raw image. Performance is computed on the PASCAL VOC 2007 dataset. Note that our method is three orders of magnitude faster than that of the original implementation. HSC [106] is slow because it uses an experimental set of features that is different than HOG. The method by Yan et al. [110] is not included in the table as its running time (0.22s per category) is reported on a single core. The methods in this table run 20 categories on six cores.

Method	aero	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP	time
Exemplar [**?**]	.19	.47	.03	.11	.09	.39	.40	.02	.06	.15	.07	.02	.44	.38	.13	.05	.20	.12	.36	.28	0.198	13.7ms
Ours	.18	.47	.03	.11	.09	.39	.40	.02	.06	.15	.07	.02	.44	.38	.13	.05	.20	.12	.36	.28	0.197	1.7ms

Table 4.4: Comparison of our method with two baselines on PASCAL VOC 2007. The two rows compare the performance of our exemplar SVM implementation with the baseline. For the top three rows running time refers to per (image$\times$category) time. For the two bottom rows running time refers to the average time to evaluate a single exemplar (that includes MATLAB overhead).

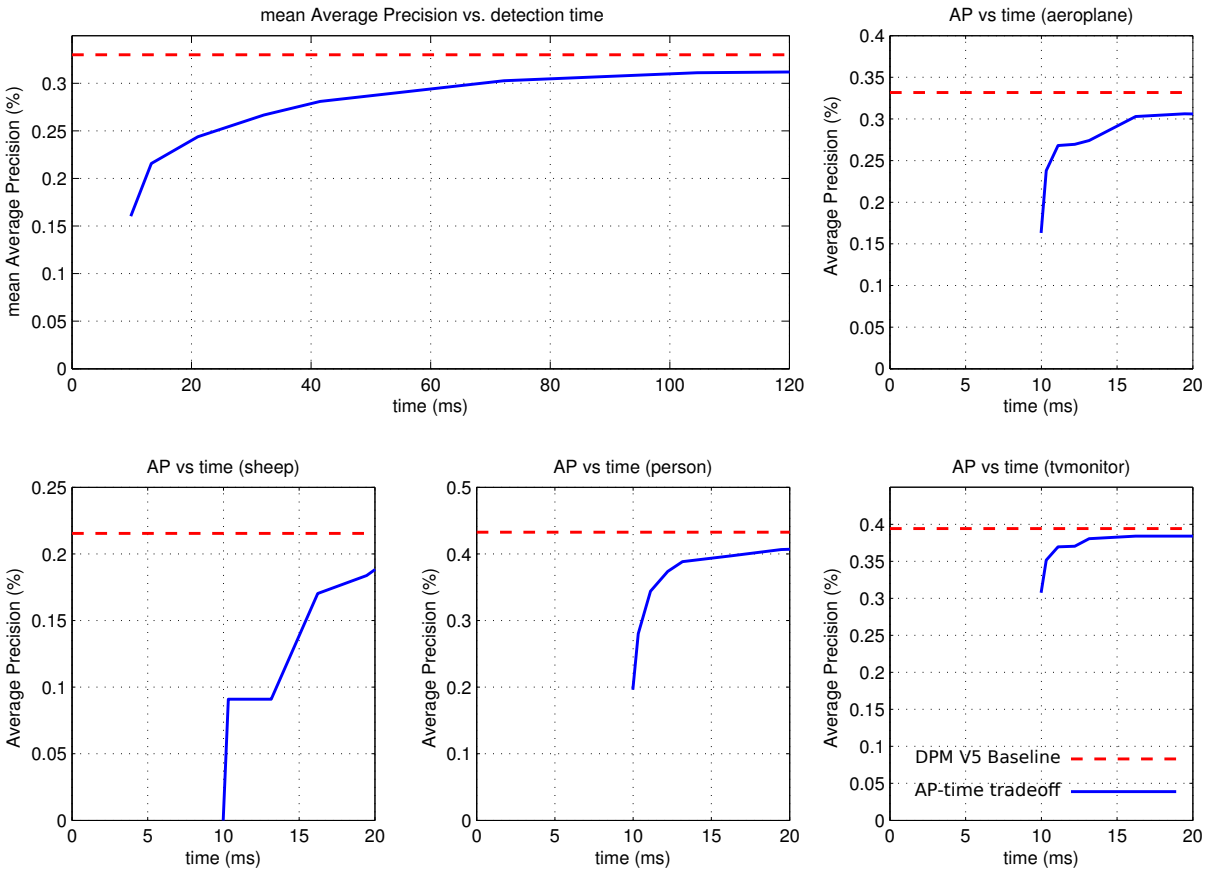


Figure 4.9: Our method operates within a time limit specified by the user. It can jointly detect the entire set of PASCAL VOC challenge categories in about 10ms, that is about 0.5ms per category. The top-left plot shows the trade-off between operation time-frame and mean Average Precision (mAP) of the 20 PASCAL categories. In this setting all 20 objects are detected jointly within the time-frame. The rest of the plots show that this trade-off for detecting a single category. In this setting only one category is detected within the time-frame. Note that different categories respond differently to the time-limit. The Sheep detector fails at 100fps while the tvmonitor detector remains robust. The red dashed line shows DPM V5 [108] baseline while the solid blue curve shows Average Precision vs. time trade-off.

Chapter 5

Performance Evaluation for Vector Quantization

In machine learning and computer vision, low precision arithmetic (e.g. 8-bit fixed-point) is enough for many kinds of computation. There is often a trade-off between precision versus speed; high precision comes at some cost on performance while low precision can provide opportunity for higher performance. Today's computers provide several choices for arithmetic precision to help maximize performance.

Even though hardware covers a wide range of choices in precision-performance trade-off, some cases are only handled by software. In this paper we study dot product and convolution that are becoming increasingly dominant computational bottleneck in machine learning applications. In these cases, the result of a computation matters only if it falls within a specified range, often, greater than zero or a different threshold. In most cases the outcome is expected to fall outside the specified range, therefore, if full precision is computed for all operations, most of the computation is going to be unused.

In convolutional neural networks *Rectified Linear Unit* (ReLU) plays an important role. RelU is defined as follows:

$$\begin{aligned} Y &\leftarrow AX^T \\ S &\leftarrow \max(Y, 0) \end{aligned} \tag{5.1}$$

where $X_{(1\times n)}$ and $Y_{(1\times m)}$ are dense vectors and $A_{(m\times n)}$ is a dense matrix. Assume we know that most elements of Y are going to be negative, therefore, S is going to be a sparse vector (with $\frac{1}{s}$ sparsity). Assume S is the final outcome and we do not use Y.

This kind of computation is the dominant computation in many computer vision and machine learning applications where numerous hypotheses are tested but most of them are unlikely to turn out true. This hypothesis testing can manifest itself in the form of convolution, matrix product, or tree search. Today in most cases, Y is computed with full precision and then positive values are extracted. In order to improve performance, one could compute Y with a lower precision, determine positive elements, and then re-estimate positive values of Y. A number of prior works demonstrate that getting rid of unnecessary computation has lead to two orders of magnitude speed-up with limited loss of accuracy [104, 4]. These techniques first estimate Y in a crude way and re-estimate select values if/when needed.

Several estimation algorithms are proposed in the literature. We review some of these techniques in Section **??**. Then we investigate vector quantization and look-up tables in Section 5.2. In this section we study several designs of look-up table and their effects on performance. In section 5.3 we investigate algorithms to evaluate look-up tables. In section 5.4 we study different designs and compare them in terms of speed, memory requirement and accuracy.

5.1 Estimation Strategies

?? In equation 5.1, if the negative elements of Y were known in advance they could be excluded from computation to gain s times speed-up. However, the problem is that Y is not known before performing the $\Theta(mn)$ matrix-vector product. An immediate solution is to first compute Y and S with lower precision (*first estimation*), then identify non-zero elements in S, and finally repeat the computation for the non-zero elements of S (high precision *re-estimation*).

In this setting, if the representation of the numerical values is compressed by a factor of c, computation time for the first estimation would be $\Theta(\frac{mn}{s})$. Similarly, the computation time for re-estimation would be $\Theta(\frac{mn}{c})$. Total computation time would be: $\Theta((\frac{1}{c} + \frac{1}{s})mn)$. In practice, s (inverse sparsity ratio) ranges from 10 to 10^5 and c (compression ratio) ranges from 1 to 8.

Therefore, in most applications running time is dominated by the first estimation stage. In order to improve overall performance, one needs to maximize c (compression ratio).

Several techniques are employed to maximize c. These techniques often come with some costs:

1. **Estimation Error**: The compression of representation often comes with some error. To deal with error, one could either ignore the error, use a conservative margin, or perform a re-estimation.

2. **Two Stage Time**: Re-estimating values involves processing the matrix twice. Therefore, the total running time depends on the sum of running times in both stages. Therefore, c and s both need to be large. In certain problems one could totally avoid the second stage and use the estimated values instead. In this case, one must consider error implications in the rest of the software.

3. **Increased Memory Requirement**: In most compression techniques the original data is left intact and a new copy is produced with a different representation. Even though the second representation can be more compact, it is still some overhead memory that comes at some cost for certain systems.

4. **Pre-computation Cost**: Some estimation techniques involve significant pre-computation cost. Therefore, the feasibility of an estimation technique depends on problem parameters. For example, if a matrix A is going to be used once, because the cost of compression is at least as large of $\theta(mn)$ compression is not going to be helpful. In most applications, however, matrix A is reused several times with various candidates for X.

The following sections will describe a number of techniques that are most commonly used.

5.1.1 Bandwidth Compression

A double or single floating point is replaced with a 16-bit or an 8-bit fixed-point. This technique [97] [90] usually yields a low compression ratio. However, first estimation is often accurate enough that a second round is not going to be needed. Using this techniques requires careful measures in order to use the available bits efficiently.

5.1.2 Sub-Byte Bandwidth Compression

In some applications [94] sub-byte bandwidth arithmetic (e.g. 3-bit $\times$ 3-bit) is enough. Processors do not support $3b \times 3b$ computation. However, collated representation of numerics and the use of POPCNT operation can efficiently simulate small sub-byte arithmetic. This technique could yield a compression c of as high as 10.

5.1.3 Principal Component Analysis

In PCA [89] the dimensionality of vectors are reduced by a factor of c. This in turn reduces the computation time of dot-product operations. PCA can provide a large compression c. However, it comes with some drawbacks. First, dimensionality reduction requires a pre-computation that must be taken into account in the performance model. Second, high compression comes with some error. In Equation 5.1 where $S \leftarrow \max(Y, 0)$ is considered, a lower threshold than 0 must be used due to the expected error. Another implication is that one needs to pre-estimate expected error. One advantage of PCA comparing to the previous techniques is that compression ratio c can be chosen from a wide spectrum. Furthermore, one can perform a two-stage pre-estimation []. Also, PCA can be used in combination with the previous techniques.

5.1.4 Vector Quantization

In Vector Quantization, the dimensions of vector B are partitioned into several groups **??**. Then each group of dimensions is quantized into a particular index. Finally the original vector is represented as an array of indices.

For example, given a 1024 dimensional vector B, it is first divided into $l = 64$ vectors $B_1, \ldots, B_{64}$ each $d = 16$ dimensions long. Then each vector B_i is quantized into one of $k = 256$ clusters to produce an index q_i. Finally, a string $Q = (q_1, \ldots, q_l)$ is considered the vector quantization of B. In practice, a floating-point vector B taking 4096 Bytes of space is compressed into 64 Bytes that is a 64 times compression. This compression comes with some computation-cost and some error. The trade-off between compression ratio, compression time, and compression error is defined by the choices of l and k.

Vector Quantization was first introduced in the 1950s as a signal compression framework [105]. In 1980s and 1990 several efficient compression algorithms based on VQ were introduced for images, audio and video signals. These algorithms were primarily introduced for compression. However, since 2010 several algorithms employed VQ primarily for computational speed-up [104, 4].

Given two quantized vectors P and Q each with length l, there are two approaches to compute their dot product:

- Reconstruct the original non-quantized vectors and perform dot-product.
- For each of the l pieces of P and Q Look up partial dot-products from a *look-up table* and add them up.

Both approaches produce similar numerical outputs. The first approach does not improve computation performance as it requires the full arithmetic in addition to reconstruction. However, the second approach can significantly reduce computation and improve performance. There are vari-

ous subtitles in this process that section 5.2 will cover. Comparing to PCA, VQ is similar as both compress a vector as a whole rather than compressing each dimension independently. However, VQ often yields a better performance-accuracy trade-off comparing to PCA (Figure xxx).

5.2 Look-up Tables to Circumvent Computation

Look-up tables can store pre-computed results for certain atomic computations. Given a computational problem, one could circumvent computation by the computation into pre-defined atomic operations. The result of these computations can be immediately accessed in the look-up table.

In a simple case, assume m vectors $B_1, \ldots, B_m$ are given in a d-dimensional space. We need to quickly approximate the dot product between every pair of vectors. Assume we have a dictionary Σ containing k vectors $U_1, \ldots, U_k$ in the d-dimensional space. The first step is to build a lookup table LUT where $\mathrm{LUT}(i, j)$ stores the dot products between every pair of U_i and U_j where $1 \leq i, j \leq k$. This step is considered a pre-process because the lookup table is built before queries are available. For each vector B_i we first find q_i to minimize $||B_i - U_{q_i}||_2$. We refer q_i as the quantization of B_i. Finally, in order to approximate the dot product between B_i and B_j we just look up the table using q_i and q_j.

5.2.1 Quantization in Variable Spaces

In the previous example the lookup table $\mathrm{LUT}(i, j)$ both i and j index the same dictionary Σ. In a more general case i and j could index different dictionaries Σ_1 and Σ_2 that could be different in size. This can be specially handy if the two vectors B_1 and B_2 to be multiplied come from different distributions in space.

If the two vectors B_1 and B_2 come from an ld-dimensional space they could be be broken up into l pieces of d-dimensions each. Then each piece could be represented with a unique dictionary.

More dictionaries require more pre-computation, more space and more cache access. However, because specialized dictionaries can minimize representation error, a specialized dictionary may need fewer elements to reproduce the same error behaviour.

5.2.2 Symmetric versus Asymmetric quantization

In the previous examples each look-up table $\mathrm{LUT}(i, j)$ is index with two indices. This is called *Symmetric Quantization* because both vectors are quantized. In certain applications, a look-up table with a single index is more useful. In dot product operations one of the vectors could be always fixed. For example a feature vector could be variable while the weight vector is fixed. In cases like this a one dimensional look-up table is used where only the vector quantization of the feature vector is indexed (*Asymmetric Quantization*).

In asymmetric quantization each look-up tables is constructed according to one weight vector. Asymmetric quantization has a few advantages and a few disadvantages over symmetric quantization. The advantages of asymmetric quantization are:

1. **Smaller error**: Because only one side is quantized, weight vector is encoded accurately so given a

2. **Packed Lookup**: If several dot product queries are always made together their corresponding look-up tables could be packed together so that multiple look-up tables could be index in one pass. This could make a better use of a cache line and improve memory bandwidth.

Symmetric Quantization has several advantages to Asymmetric quantization:

1. **Light Pre-processing**: In symmetric look-up tables the pre-computed symmetric look-up table is enough for all weight vectors that could be introduced in the future. In contrast, in asymmetric look-up tables each new weight vector needs a separate look-up table to be processed.

2. **Lower memory requirements**: If there are many weight vectors to be processed, storing an asymmetric look-up table for each weight could be memory intensive. Moreover, if they do not fit into cache all together, cache bandwidth could become a significant limiting factor.

As a rule of thumb, If there are tens of thousands of weight vectors or more asymmetric quantization could face memory and bandwidth limitations so symmetric quantization would be needed.

5.3 Fast Look-up Algorithms

Vector Quantization can be very fast, however, speed-up depends on several factors including:

- The right choice of algorithm. Up to this point a few algorithms are discussed. In this section we will explain the algorithms in more details.
- The choice of parameters. Parameters include k (dictionary size), l (template size), m the number of templates.
- Implementation. A good implementation must allow for parallel processing, use a memory access pattern that utilized cache, and employ SIMD operations for further parallelization.

All of these factors need to be optimized at the same time. Further, in order to use the fastest implementation they need to be compatible. We have implemented three algorithms for vector quantization and template evaluation. We will discuss them in this section along with the baseline.

5.3.1 Baseline Algorithm

We use matrix multiplication as baseline. This baseline is exact and we expect our approximations to be faster than the baseline. We use MATLAB multiplication implementation that uses BLAS and LAPACK. BLAS and LAPACK are very optimized linear algebra and matrix libraries that

implement matrix operations in parallel. MATLAB also uses Strassen algorithm whenever it helps speed up computation. In our case MATLAB definitely uses Strassen algorithm as we multiply two very large matrices at once.

We have a total of 625 configurations that include five choices for k (dictionary size), five choices for l (template size), five choices for m (the number of templates) and five choices of number of threads. The number of feature vectors to be evaluated n is selected accordingly to ensure the total number of float operations remains at 3×10^{10}. This is to be able to remove the effect of problem size and focus on the effect of individual parameters. We use single floating point operations for all algorithms. Our baseline takes 0.712 seconds to complete.

5.3.2 Symmetric Quantization

Symmetric Quantization quantizes both weight vectors and feature vectors to a dictionary. Since Weight vectors and feature vectors are both quantized the size of look-up table neither depend on the number of templates nor the number of feature vectors. Instead, it quadratically depends on k.

Our algorithm is implemented using OpenMP in a way that it could effectively utilize multiple cores (Figure 5.1).

5.3.3 Asymmetric Quantization - Slow

Asymmetric quantization initializes a look-up table for each partition in each template. Memory requirement for look-up tables is $4mlk$ Bytes. In our experiments this can go as high as 1GB. Access to such a large table in an irregular fashion could be very time consuming and inefficient. This is why without a proper data-structure vector quantization cannot compete with the baseline. We use two implementations of asymmetric vector quantization for comparison.

Memory access pattern for look-up tables is irregular by default. This means that it is not predictable which memory address is going to be accessed. As a result, the time to access memory

dominates computation time. This affects the speed dynamics.

5.3.4 Asymmetric Quantization - Fast

The fast implementation of Asymmetric quantization uses two techniques to achieve further speedup.

1. packed look-up tables. Because multiple templates need to be evaluated for each input feature vector, one could pack them and index them together. In this implementation, eight look-up tables corresponding to eight different templates are packed together and accessed at once. This has a few implications. First, the eight look-up tables effectively become one look-up table so the overhead to find the address and load the values is minimized. Second, a 64-byte cache line could be utilized efficiently.

2. SIMD operations. We use AVX operations to add up partial dot products that are extracted from look-up tables. AVX operations are advanced versions of SIMD operations that can handle 512-bits or 8 floating point variables. AVX operations as fast as regular SISD operations. However, they have eight times higher throughput.

The effects of these features are studied in the next section.

5.4 Experimental Results

We use an Intel Xeon E5-1650 processor with 2.8GHz clock frequency. This processor has six cores each with a 256KB L2 cache. L3 is is 15MB. Total memory is 48GB and the memory bandwidth is 10GB/s.

We compare three algorithms that are described in the previous section: Symmetric Quantization, Asymmetric Quantization - slow, and Asymmetric Quantization - fast. All of these algorithms are implemented in parallel using OpenMP. These algorithms are tested using 1 to 12 threads and

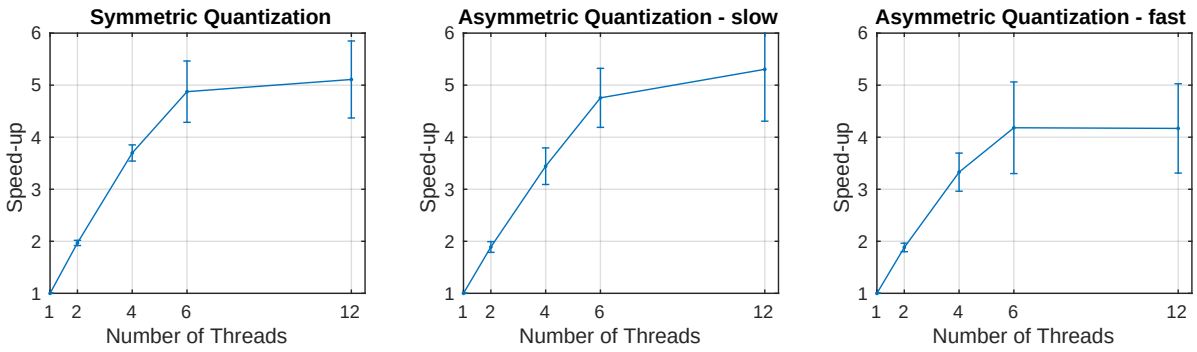


Figure 5.1: The effect of the number of threads on speed-up. The computer that the benchmark is evaluated has 6 cores. We compared 1, 2, 4, 6, 12 threads. All algorithms are implemented using openMP to take advantage of multiple cores as much as possible. Speed-up is computed for different number of threads over 105 configurations of m, k, and l. Then the speed-up factors are averaged and confidence intervals are extracted and visualized. **Symmetric Quantization** and **Asymmetric Quantization - slow** can speed up by a factor of 5 using 6 cores because all threads run independently. **Asymmetric Quantization - fast** goes up to 4 times speed-up at most. We believe this is because we use AVX operations. AVX operations use the full width of ALUs in processors. Therefore, hyper threading is not possible when AVX operations are in use. In the presence of other processes AVX threads need to completely shut down and resume in context switches. An alternative is to avoid using AVX operations and using SSE4.2 operations instead. Although this alternatives allows for hyper threading (more than 6 threads on a 6 core processor), throughput is divided by half so the net effect is negative.

they utilize cores effectively (Figure 5.1).

We have a total of 625 configurations that include five choices for k (dictionary size), five choices for l (template size), five choices for m (the number of templates) and five choices of number of threads. The number of feature vectors to be evaluated n is selected accordingly to ensure the total number of float operations remains at 3×10^{10}. This is to be able to remove the effect of problem size and focus on the effect of individual parameters. We use single floating point operations for all algorithms. Our baseline takes 0.712 seconds to complete.

Perhaps the most important factor to decide on is dictionary size k. Dictionary size have distinct effect on different implementations. For Symmetric Quantization a small dictionary size has no effect on performance but a large dictionary size has a large effect. For Assymetric Quantization dictionary size affects memory requirements and thus speed (Figure 5.2).

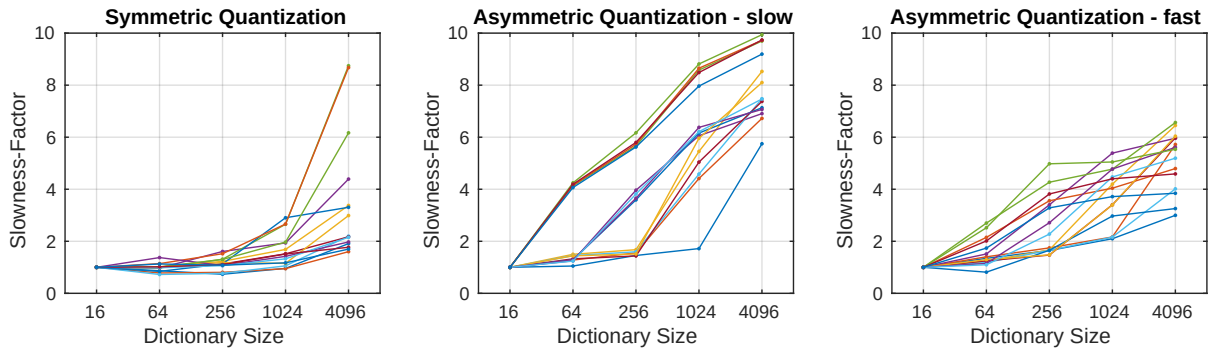


Figure 5.2: The effect of dictionary size (referred to as k) on speed for the three different algorithms. For simplicity, we use slowness factor (inverse speed-up) instead of speed-up. In this figure dictionaries of sizes 16 to 4096 are compared. The effects of increasing dictionary size is illustrated using fourteen different configurations for other variables. This gives a better understanding of dictionary size effects. **Symmetric Quantization**: This algorithm uses $\Theta(k^2)$ memory. As a result, memory use increases quadratically. For $K \leq 256$ look-up table takes at most 1MB of memory. For $k = 1024$ look-up table can overflow the memory depending of the number of threads used (1 to 12). for higher number of threads it is more likely to overflow. For $k = 4096$ it is more likely to overflow. Therefore running time in most cases increases by a factor of about 2. **Asymmetric Quantization - slow**: Memory requirement for this algorithm can be as little as 16KB or as much as 1GB depending on m, l and k. For most of configurations in this experiment there is one threshold that triggers running time to suddenly increase. This is because memory access patterns in the slow algorithm is irregular and proper caching depends on table size. **Asymmetric Quantization - fast**: In contrast, the running time of this algorithm is not controlled by cache requirements because the fast algorithm uses cache more effectively by packing lookup tables and accessing multiple look-up tables at the same time.

The effect of the length of quantized weight vectors on speed is illustrated in Figure 5.3. In this figure weight vectors of length 16 to 4096 are compared. The effects of variable weight vector lengths is compared using 105 different configurations for other variables. All configurations are set up in a way to ensure the total number of float operations are equal to 3.4×10^{10} for a fair comparison. This means when weight vectors are short more feature vectors are evaluated and when weight vectors are long fewer feature vectors are evaluated. This is to ensure the number of float operations remain 3.4×10^{10}.

The effect of the number of templates on speed for the three different algorithms is shown in Figure 5.4. Look-up tables have different memory requirements depending on m, k, and l

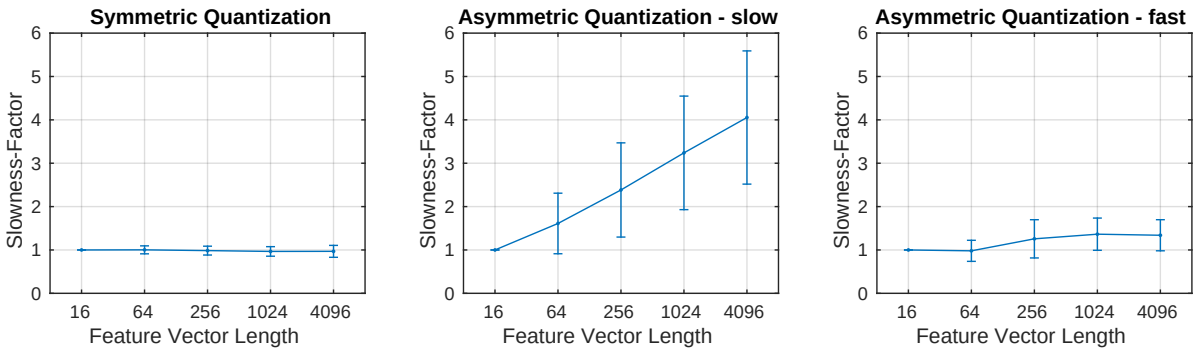


Figure 5.3: The effect of the length of quantized weight vectors (length referred to as l that equals the number of times a look-up table needs to be accessed for one feature vector) on speed for the three different algorithms. For simplicity, we use slowness factor (inverse speed-up) instead of speed-up. In this figure weight vectors of length 16 to 4096 are compared. The effects of variable weight vector lengths is compared using 105 different configurations for other variables. Then error bars are computed and a single curve is shown for simplicity. All configurations are set up in a way to ensure the total number of float operations are equal to 3.4×10^{10} for a fair comparison. This means when weight vectors are short more feature vectors are evaluated and when weight vectors are long fewer feature vectors are evaluated. This is to ensure the number of float operations remain 3.4×10^{10}. **Symmetric Quantization**: The speed of this algorithm does not depend on the length of feature vectors. In fact the longer the feature vectors, the fewer partial dot products that are required to be stored. **Asymmetric Quantization - slow**: Memory requirements for look-up tables linearly depends on the length of weight vectors. Therefore by increasing the length of the feature vectors more memory is required. Increased memory requirements causes cache bandwidth to saturate and control running time. **Asymmetric Quantization - fast**: Total memory requirements for this algorithm is similar to the slow algorithm. However, in the fast algorithm cache access is improved for two reasons. 1- Look-up tables are loaded step by step. Therefore the number of cache miss is minimized. 2- Look-up tables are packed in a way that multiple look-up tables can be accessed by one look up. In fact 16 look-up tables are read at one in order to maximize the utilization of cache lines.

as well as the kind of algorithm used. Increasing the number of templates m does not increase memory requirements for symmetric quantization. However, Memory requirement for look-up tables linearly depends on m. Therefore by increasing the number of templates to be evaluated both slow and fast algorithms are affected.

Ultimately, memory usage has the most apparent effect on speed (Figure 5.5). Memory depends on the choice of algorithm, m, k, and l. Memory requirements can range from 16KB to 1GB. We avoided larger memories as a different addressing scheme would be required for larger

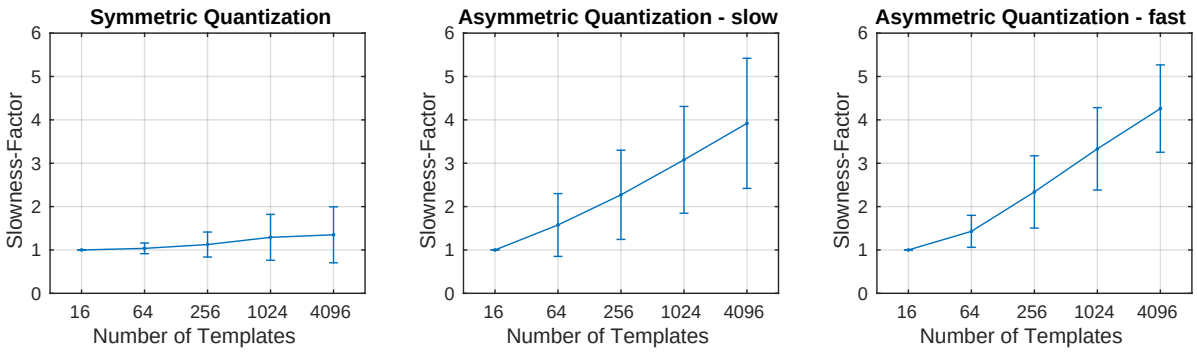


Figure 5.4: The effect of the number of templates (weight vectors) on speed for the three different algorithms. Look-up tables have different memory requirements depending on m, k, and l as well as the kind of algorithm used. **Symmetric Quantization**: Increasing the number of templates m does not increase memory requirements for symmetric quantization. As a result, increasing the number of templates does not significantly affect speed. **Asymmetric Quantization - slow and fast**: Memory requirement for look-up tables linearly depends on m. Therefore by increasing the number of templates to be evaluated both slow and fast algorithms are affected. The fast algorithm is not any more scalable than the slow algorithm due to the way look-up tables are stored in memory. The fast algorithm packs multiple look-up tables belonging to different templates. Therefore by increasing the number of templates the number of packs increase. Evaluating a pack of templates completely exhausts cache therefore all templates need to frequently be loaded and unloaded from cache.

memories. Settings are designed to ensure equal number of floating point operations.

In practice, parameters are chosen according to the trade-off between error and speed-up. Figure 5.6 illustrates speedup versus relative error. Speed-up is calculated according to MATLAB baseline that uses BLAS and LAPACK. Several factors can affect this trade-off including k (dictionary size), l (template size). We use a variety of different configurations to illustrate the effect of parameters on the trade-off. For Symmetric Quantization, $k = 16$ to $k = 256$ provide the best working point. For Asymmetric Quantization $k = 16$ provides the best working point. Figure 5.6 clearly shows that Asymmetric quantization is superior in terms of speed only if proper data structure is used and cache is handled effectively.

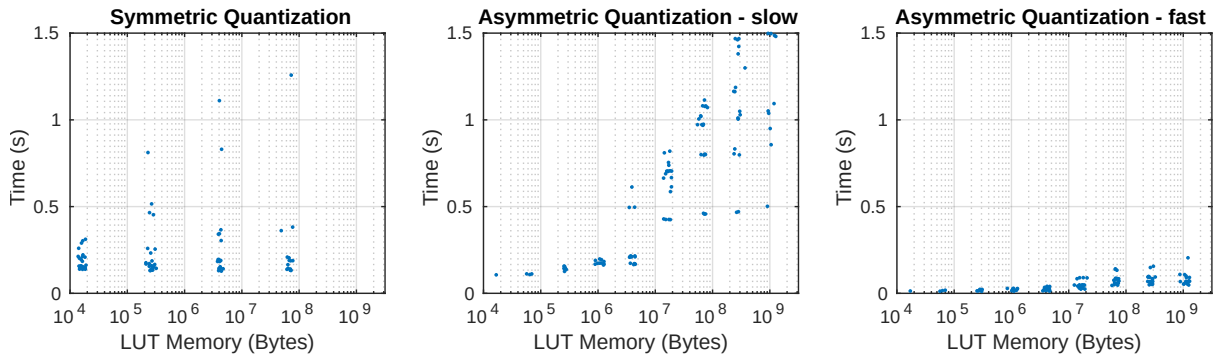


Figure 5.5: For each setting, storing look-up tables requires certain amount of memory depending on m, k, and l. Memory requirements can range from 16KB to 1GB. We avoided larger memories as a different addressing scheme would be required for larger memories. Settings are designed to ensure equal number of floating point operations. Therefore, the residual effect highly depends on memory size and access pattern. **Symmetric Quantization**: This algorithm requires at most 64MB of memory for a 4096×4096 look-up table. Therefore, memory requirement for look-up tables is limited and is not a limiting factor. As shown in this graph detection time depends more on factors other than memory requirements. **Asymmetric Quantization - slow**: There is clear jump on and after $4MB$ memory use. We believe this is due to cache overflow. The processor in use has 15MB of cache. In this visualization only runs with 6 cores are considered because variation in the number of cores produces a confusing variation in time. **Asymmetric Quantization - fast**: This algorithm has a more regular memory access pattern by grouping look-up tables and accessing multiple of them in one round. The fast algorithm uses AVX operations to directly add the variables in a SIMD style operation. 16 floating points are accessed and processed at the same time effectively treating them as 512-bit variables.

5.5 Discussion

In a wide range of applications including convolutional neural networks, the exact value of a function matters only if it falls within a range. Conventional architecture computes the full accuracy at once which is most likely not going to be needed later on. One could first compute a rough estimate, and continue to the rough estimate only if the value falls in a range. We studied dot product as a simple operation to show this concept.

Vector quantization can be used to estimate dot-product quickly. We discussed a few schemes for vector quantization with different characteristics and implications. We studied Symmetric Quantization and Asymmetric Quantization. For Asymmetric Quantization we compared two dis-

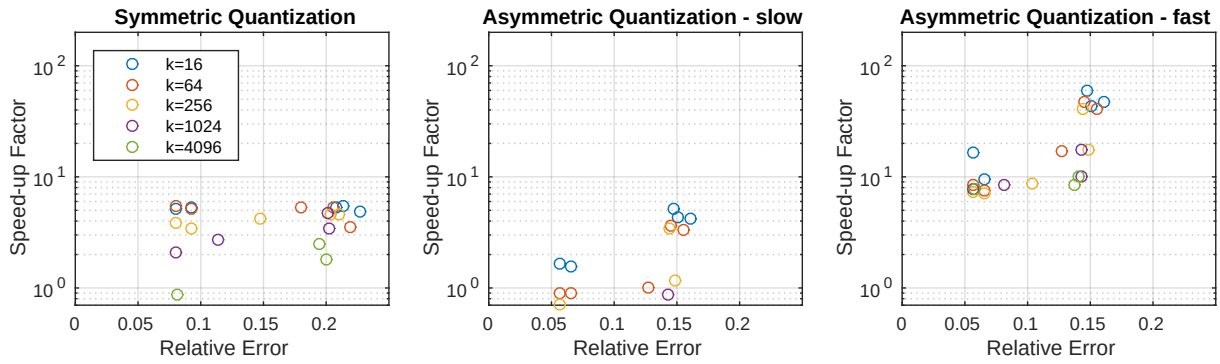


Figure 5.6: The trade-off between error and speed-up. Several factors can affect this trade-off including k (dictionary size), l (template size), and the choice of algorithm. Baseline for speed-up is MATLAB's matrix multiplication that is implemented using BLAS and LAPACK. This baseline uses six cores and is among the most efficient matrix multiplication algorithms that uses Strassen algorithm for further speed-up. Here we have compared 21 different configurations to build a reliable visualization. The colors in the plots refer to k. **Symmetric Quantization**: This algorithm quantizes both weight vectors and feature vectors. Therefore, it has a higher error comparing to asymmetric quantization. Please note that $k = 4096$ and $k = 1024$ lead to lower speed-up factors (sometimes less than 1). This is because the two dimensional look-up table cannot fit in cache. Since all the access to the look-up table is irregular, cache is not used effectively. There is not a major speed-up distinction between $k = 256$, $k = 64$ and $k = 16$. However, $k = 256$ provides a lower error. This is because the latter three sizes can fit in the cache. **Asymmetric Quantization - slow**: Here the effect of look-up table size on speed-up is more apparent. Notice that $k = 16$ is significantly faster than other cases. Here the speed is defined by proper cache utilization. The speed-up for $k = 1024$ and $k = 2048$ is often lower than 1. **Asymmetric Quantization - fast**: This algorithm is about an order of magnitude faster than the slow version. Please note that $k = 1024$ and $k = 2048$ perform the worst in the trade-off. The conclusion for this plot is that the full benefit of look-up tables is only available if they can fit in cache and properly accessed.

tinct implementations. Each of these implementations can take a range of configurations. We compared our implementation in various configurations and showed that Symmetric Quantization can work best only if the right choice of parameters and the right implementation are used.

We evaluated the combined effects of five variables on speed, memory and running time. In more specific applications one could introduce, control and study further variables. We provided these experiments to illustrate the role of different parameters and general guidelines for the right choice of parameters for vector quantization.

Chapter 6

Conclusion

The goal of computer vision is to understand images accurately, fast and with great details. Some vision techniques focus on accuracy but they are slow. Some techniques are fast but they trade off accuracy and detail. Some techniques provide great details but they are slow and sometimes inaccurate. We explored these three goals and studied the dynamics between them. We also studied strategies to improve all of the three criteria together.

Improving details often requires more sophisticated modeling. We studied details at two levels. First we proposed an algorithm to describe images with sentences. We represent both images and sentences in a joint space and use search for relevant matches. We showed that individual objects are not necessarily the best things to detect. Detecting composites of objects can be more accurate, and more informative at the same time. We further investigated visual phrases in different details.

More details often come with more computation because it requires more detailed computation. More details also often comes with lower accuracy because every new detail comes with its own chance of error. We showed that improved details eventually requires more systematic modeling. We also show that it is important to choose the right degree of output detail opportunistically.

Improving accuracy also often comes with more computation. In order to break out of this trade-off we studied techniques for fast object detection. We showed that vector quantization is an effective technique for speed-up. We also study fast object detection from an engineering perspective. We argued that a desirable object detector must: 1- be able to work with legacy templates, 2- be random access, 3- be able to trade accuracy versus speed, 4- have any-time property. We developed a technique to have all of these features together while being fast. We apply these techniques

to deformable parts model object detectors and show two orders of magnitude speed-up. We finally investigated the consequences of this architecture with a view of improving convolutional neural networks.

References

[1] A. Farhadi and M. Hejrati and M. A. Sadeghi and P. Young1 and C. Rashtchian1 and J. Hockenmaier1 and D. Forsyth Every Picture Tells a Story: Generating Sentences from Images In *European Conference on Computer Vision (ECCV)*, 2010.

[2] M. A. Sadeghi, and A. Farhadi, Recognition using Visual Phrases, In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011

[3] M. A. Sadeghi and D. Forsyth Phrasal Recognition In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.

[4] M. A. Sadeghi and D. Forsyth Fast Template Evaluation with Vector Quantization In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[5] M. A. Sadeghi and D. Forsyth 30Hz Object Detection with DPM V5 In *European Conference on Computer Vision (ECCV)*, 2014.

[6] Barnard, K., Duygulu, P., Forsyth, D.: Clustering art. In: CVPR. (2001) II:434–441

[7] Mori, Y., Takahashi, H., Oka, R.: Image-to-word transformation based on dividing and vector quantizing images with words. In: WMISR. (1999)

[8] Duygulu, P., Barnard, K., de Freitas, N., Forsyth, D.: Object recognition as machine translation. In: ECCV. (2002) IV: 97–112

[9] Datta, R., Li, J., Wang, J.Z.: Content-based image retrieval: approaches and trends of the new age. In: MIR '05. (2005) 253–262

[10] Forsyth, D., Berg, T., Alm, C., Farhadi, A., Hockenmaier, J., Loeff, N., Wang, G.: Words and pictures: Categories, modifiers, depiction and iconography. In: Object Categorization: Computer and Human Vision Perspectives, CUP (2009)

[11] Phillips, P.J., Newton, E.: Meta-analysis of face recognition algorithms. In: ICAFGR. (2002)

[12] Gupta, A., Davis, L.: Beyond nouns: Exploiting prepositions and comparative adjectives for learning visual classifiers. In: ECCV. (2008)

[13] Li, L.J., Fei-Fei, L.: What, where and who? classifying event by scene and object recognition. In: ICCV. (2007)

[14] Li, L.J., Socher, R., Fei-Fei, L.: Towards total scene understanding:classification, annotation and segmentation in an automatic framework. In: CVPR. (2009)

[15] Gupta, A., Davis, L.: Objects in action: An approach for combining action understanding and object perception. In: CVPR. (2007)

[16] A. Gupta, A.K., Davis, L.: Observing human-object interactions: Using spatial and functional compatibility for recognition. In: Trans on PAMI. (2009)

[17] Yao, B., Fei-Fei, L.: Modeling mutual context of object and human pose in human-object interaction activities. In: CVPR. (2010)

[18] Berg, T.L., Berg, A.C., Edwards, J., Forsyth, D.A.: Who's in the picture. In: Advances in Neural Information Processing. (2004)

[19] Mensink, T., Verbeek, J.: Improving people search using query expansions: How friends help to find people. In: ECCV. (2008)

[20] Luo, J., Caputo, B., Ferrari, V.: Who's doing what: Joint modeling of names and verbs for simultaneous face and pose annotation. In: NIPS. (2009)

[21] Coyne, B., Sproat, R.: Wordseye: an automatic text-to-scene conversion system. In: SIGGRAPH '01. (2001)

[22] Gupta, A., Srinivasan, P., Shi, J., Davis, L.: Understanding videos, constructing plots: Learning a visually grounded storyline model from annotated videos. In: CVPR. (2009)

[23] Yao, B.Z., Yang, X., Lin, L., Lee, M.W., Zhu, S.C.: I2t: Image parsing to text description. Proc. IEEE (2010) In Press.

[24] Felzenszwalb, P., Mcallester, D., Ramanan, D.: A discriminatively trained, multiscale, deformable part model. CVPR 2008 (2008)

[25] Hoiem, D., Divvala, S., Hays, J.: Pascal voc 2009 challenge. In: PASCAL challenge workshop in ECCV. (2009)

[26] Oliva, A., Torralba, A.: Building the gist of a scene: the role of global image features in recognition. In: Progress in Brain Research. (2006) 2006

[27] Curran, J., Clark, S., Bos, J.: Linguistically motivated large-scale nlp with c&c and boxer. (In: ACL) 33–36

[28] Lin, D.: An information-theoretic definition of similarity. In: ICML. (1998) 296–304

[29] Taskar, B., Chatalbashev, V., Koller, D., Guestrin, C.: Learning structured prediction models: a large margin approach. In: ICML. (2005) 896–903

[30] Ratliff, N., Bagnell, J.A., Zinkevich, M.: Subgradient methods for maximum margin structured learning. In: ICML. (2006)

[31] Rashtchian, C., Young, P., Hodosh, M., Hockenmaier, J.: Collecting image annotations using amazon's mechanical turk. (In: NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk)

[32] Premraj, V. ; Ordonez, V. ; Dhar, S. ; Siming Li ; Yejin Choi ; Berg, A.C. ; Berg, T.L. BabyTalk: Understanding and Generating Simple Image Descriptions *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2013.

[33] V. Ordonez and G. Kulkarni and T. L. Berg Im2Text: Describing Images Using 1 Million Captioned Photographs *Advances in Neural Information Processing Systems 24*, 2011.

[34] Y. Yang and C. L. Teo and H. Daum and Y. Aloimonos Corpus-guided sentence generation of natural images *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011.

[35] M. Mitchell and X. Han and J. Dodge and A. Mensch and A.. Goyal and A. Berg and K. Yamaguchi and T. Berg and K. Stratos Midge: Generating Image Descriptions from Computer Vision Detections *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, 2012.

[36] M. Hodosh and P. Young and J. Hockenmaier Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics *Journal of Artificial Intelligence Research*, 2013.

[37] P. Young and A. Lai and M. Hodosh and J. Hockenmaier From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions *Transactions of the Association for Computational Linguistics*, 2014.

[38] P. Das and C. Xu and R. F. Doell and J. J. Corso A Thousand Frames in Just a Few Words: Lingual Description of Videos through Latent Topics and Sparse Object Stitching *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[39] A. Karpathy and F. Li Deep Visual-Semantic Alignments for Generating Image Descriptions *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[40] R. Socher and A. Karpathy and Q. V. Le and C. D. Manning and A. Y. Ng Grounded Compositional Semantics for Finding and Describing Images with Sentences *Transactions of the Association for Computational Linguistics*, 2014.

[41] O. Vinyals and A. Toshev and S. Bengio and D. Erhan Show and Tell: A Neural Image Caption Generator *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[42] Y. Amit and A. Trouvé. Pop: Patchwork of parts models for object recognition. *Internatioanl Journal of Computer Vision*, 2007.

[43] C. Desai, D. Ramanan, and C. Fowlkes. Discriminative models for multi-class object layout. In *International Conference on Computer Vision*, 2010.

[44] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2001.

[45] J. Coughlan, A. Yuille, C. English, and D. Snow. Efficient optimization of a deformable template using dynamic programming. *IEEE Conference on Computer Vision and Pattern Recognition*, 1998.

[46] D. Crandall, P. Felzenszwalb, and D. Huttenlocher. Spatial priors for part-based recognition using statistical models. *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[47] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 2010.

[48] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester. Discriminatively trained deformable part models, release 4. http://people.cs.uchicago.edu/ pff/latent-release4/.

[49] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[50] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. *IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[51] P. Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010.

[52] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Conference on Computer Vision and Pattern Recognition*, 2006.

[53] N. Loeff and A. Farhadi. Scene discovery by matrix factorization. In *European Conference on Computer Vision*, 2008.

[54] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 2001.

[55] C. Li, D. Parikh, and T. Chen. Automatic Discovery of Groups of Objects for Scene Understanding. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[56] Y. Yang, A. Kannan, S. Baker, and D. Ramanan. Recognizing Proxemics in Personal Photos. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[57] A. Gupta, A. Kembhavi, and L. Davis. Observing human-object interactions: Using spatial and functional compatibility for recognition. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009.

[58] C. Desai, D. Ramanan, and C. Fowlkes. Discriminative models for static human-object interactions. In *CVPR Workshop on Statistical Models in Computer Vision*, 2010.

[59] W. Yang, Y. Wang, and G. Mori. Recognizing human actions from still images with latent poses. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[60] V. Delaitre, I. Laptev, and J. Sivic. Recognizing human actions in still images: A study of bag-of-features and part-based representations. In *British Machine Vision Conference*, 2010.

[61] S. Maji, L. Bourdev, and J. Malik. Action recognition from a distributed representation of pose and appearance. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[62] A. Prest, C. Schmid, and V. Ferrari. Weakly supervised learning of interaction between humans and objects. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[63] H. Pirsiavash, and D. Ramanan. Detecting activities of daily living in first-person camera views. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[64] C. Desai, and D. Ramanan. Detecting actions, poses, and objects with relational phraselets. In *European Conference on Computer Vision*, 2012.

[65] X. Li, C. Snoek, M. Worring, and A. Smeulders. Harvesting Social Images for Bi-Concept Search. In *IEEE Transactions on Multimedia*, 2012.

[66] C. Li, D. Parikh, and T. Chen. Extracting adaptive contextual cues from unlabeled regions. In *IEEE International Conference on Computer Vision*, 2011.

[67] J. Deng, A. Berg, Kai. Li, L. Fei-Fei. What Does Classifying More Than 10,000 Image Categories Tell Us? In *European Conference on Computer Vision (ECCV)*, 2010

[68] X. Zhu, C. Vondrick, D. Ramanan, C. Fowlkes. Do We Need More Training Data or Better Models for Object Detection? In *British Machine Vision Conference*, 2012.

[69] Y. Aytar, A. Zisserman. Tabula rasa: Model transfer for object category detection. In *IEEE International Conference on Computer Vision*, 2011.

[70] J. Lim, R. Salakhutdinov, A. Torralba. Transfer learning by borrowing examples for multi-class object detection. In *Conference on Neural Information Processing Systems*, 2011.

[71] D. Hoiem, Y. Chodpathumwan, Q. Dai. Diagnosing error in object detectors. In *European Conference on Computer Vision*, 2012.

[72] S. Singh, A. Gupta, and A. Efros. Unsupervised Discovery of Mid-Level Discriminative Patches. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[73] Y. Ushiku, T. Harada and Y. Kuniyoshi. Efficient Image Annotation for Automatic Sentence Generation. *Proceedings of the 20th ACM international conference on Multimedia*, 2012.

[74] F. Siyahjani, and G. Doretto. Learning a Context Aware Dictionary for Sparse Representation. *Asian Conference on Computer Vision*, 2012.

[75] H. Bilen, V. Namboodiri, and L. Van Gool. Classification with Global, Local and Shared Features. *DAGM-OAGM Joint Pattern Recognition Symposium*, 2012.

[76] Y. Feng, and M. Lapata. Automatic Caption Generation for News Images. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.

[77] S. K. Divvala, A. Efros, M. Hebert. How important are Deformable Parts in the Deformable Parts Model? In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[78] D. Park, D. Ramanan, and C. Fowlkes. Multiresolution models for object detection. In *Proceedings of the 11th European conference on Computer vision: Part IV*, 2010.

[79] L. Bourdev, S. Maji, T. Brox, and J. Malik, Detecting People Using Mutually Consistent Poselet Activations. In *European Conference on Computer Vision (ECCV)*, 2010

[80] O. Chum, and A. Zisserman. An Exemplar Model for Learning Object Classes. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007

[81] C. Gu,and X.. Ren, Discriminative mixture-of-templates for viewpoint classification In *Proceedings of the 11th European conference on Computer vision: Part V*, 2010

[82] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and Fei-Fei Li. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009

[83] T. Deselaers, and V. Ferrari. Visual and Semantic Similarity in ImageNet In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011

[84] A. Gupta, A. Kembhavi, and L. Davis. Observing Human-Object Interactions: Using Spatial and Functional Compatibility for Recognition In *IEEE Trans. Pattern Anal. Mach. Intell.*,2009

[85] B. Yao, and L. Fei-Fei Recognizing Human-Object Interactions in Still Images by Modeling the Mutual Context of Objects and Human Poses In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012

[86] W. Choi, and Y. Chao, and C. Pantofaru, and s. Savarese Understanding Indoor Scenes Using 3D Geometric Phrases In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013

[87] F. Sadeghi, and, K. Santosh, and A. Farhadi VisKE: Visual Knowledge Extraction and Question Answering by Visual Verification of Relation Phrases. In *IEEE Conference on Computer Vision and Pattern Recognition*. 2015.

[88] X. Chen, and A. Ritter, and A. Gupta, and T. Mitchell Sense Discovery via Co-Clustering on Images and Text In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[89] P. F. Felzenszwalb and R. B. Girshick and D. McAllester. Cascade Object Detection with Deformable Part Models. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[90] P. Dollár and R. Appel and S. Belongie and P. Perona Fast Feature Pyramids for Object Detection In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014.

[91] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[92] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features In *IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[93] I. Endres and D. Hoiem Category Independent Object Proposals In *European Conference on Computer Vision*, 2010.

[94] Ming. Cheng and Z. Zhang and W. Lin and P. Torr BING: Binarized Normed Gradients for Objectness Estimation at 300fps In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[95] D. Nister and H. Stewenius Scalable Recognition with a Vocabulary Tree In *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[96] H. Pirsiavash and D. Ramanan Steerable part models In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[97] P. Dollár and R. Appel and W. Kienzle Crosstalk Cascades for Frame-Rate Pedestrian Detection In *European Conference on Computer Vision*, 2012.

[98] C. Dubout and F. Fleuret. Exact Acceleration of Linear Object Detectors. In *European Conference on Computer Vision*, 2012.

[99] M. Pedersoli and J. Gonzalez and A. Bagdanov and and JJ. Villanueva. Recursive Coarse-to-Fine Localization for fast Object Detection. In *European Conference on Computer Vision*, 2010.

[100] I. Kokkinos. Bounding Part Scores for Rapid Detection with Deformable Part Models In *2nd Parts and Attributes Workshop, in conjunction with ECCV*, 2012.

[101] T. Dean and M. Ruzon and M. Segal and J. Shlens and S. Vijayanarasimhan and J. Yagnik. Fast, Accurate Detection of 100,000 Object Classes on a Single Machine. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[102] P. Indyk and R. Motwani. Approximate nearest neighbours: Towards removing the curse of dimensionality. In *ACM Symposium on Theory of Computing*, 1998.

[103] A. Vedaldi and A. Zisserman. Sparse Kernel Approximations for Efficient Classification and Detection In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[104] Herv Jgou and Matthijs Douze and Cordelia Schmid. Product quantization for nearest neighbour search. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[105] R. M. Gray and D. L. Neuhoff. Quantization. In *IEEE Transactions on Information Theory*, 1998.

[106] X. Ren and D. Ramanan. Histograms of Sparse Codes for Object Detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[107] P. Felzenszwalb and R. Girshick and D. McAllester. Discriminatively Trained Deformable Part Models, Release 4. In *http://people.cs.uchicago.edu/ pff/latent-release4/.*

[108] R. Girshick and P. Felzenszwalb and D. McAllester. Discriminatively Trained Deformable Part Models, Release 5. In *http://people.cs.uchicago.edu/ rbg/latent-release5/.*

[109] H. Song and S. Zickler and T. Althoff and R. Girshick and M. Fritz and C. Geyer, P. Felzenszwalb, T. Darrell. Sparselet Models for Efficient Multiclass Object Detection In *European Conference on Computer Vision*, 2012.

[110] J. Yan, Z. Lei and L. Wen and S. Z. Li. The Fastest Deformable Part Model for Object Detection In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[111] S. Bengio and J. Weston and D. Grangier. Label embedding trees for large multi-class tasks. In *Advances in Neural Information Processing Systems*, 2010.

[112] R. Benenson and M. Mathias and R. Timofte and L. Van Gool. Pedestrian detection at 100 frames per second. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.